Real-Time Computational Gigapixel Multi-Camera Systems

THÈSE N^{O} 6919 (2016)

PRÉSENTÉE LE 15 JANVIER 2016

À LA FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
LABORATOIRE DE SYSTÈMES MICROÉLECTRONIQUES
PROGRAMME DOCTORAL EN GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Vladan POPOVIC

acceptée sur proposition du jury:

Prof. J.-Ph. Thiran, président du jury
Prof. Y. Leblebici, directeur de thèse
Prof. G. Wetzstein, rapporteur
Dr A. Smolic, rapporteur
Prof. S. Süsstrunk , rapporteuse

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2016

The difference between the possible and the impossible
is merely a measure of man's determination.
— Captain James Thain

Acknowledgements

This thesis marks the end of a 4-year period full of hard work and learning, but also full of joy, happiness, and interesting events. The latter part is a natural consequence of being around great people, whose help, guidance, and friendship have helped me throughout my PhD.
First, I would like to express my deepest gratitude to my thesis advisor, Prof. Yusuf Leblebici, for giving me the opportunity to work at LSM. I would like to thank him for his patience, unconditional support during tough times, and belief in my abilities when even I doubted. The lessons I learned from him about both academic research and life in general are simply priceless. I would also like to thank my jury members Prof. Sabine Süsstrunk, Prof. Gordon Wetzstein, Dr. Aljoša Smolić, and Prof. Jean-Philippe Thiran for their invaluable comments about my work and this thesis. My additional gratitude goes to Gordon, who accepted me to be a visitor in his group at Stanford University, where I have definitely learned a lot in the fields slightly outside of my PhD topic. I also must thank Dr. Alain Vachoux and Dr. Alexandre Schmid for their support during my stay at LSM.
With respect to my research project, I would like to thank Armasuisse for believing in this project and for their continuous funding during these years. I would especially like to thank Dr. Peter Wellig and Dr. Beat Ott for their inputs and constructive comments.
This PhD journey would not have been a joyful ride if it had not been for the extraordinary people and great colleagues at LSM. It has been a great pleasure to work among you, and to have long and frequent coffee breaks. I would like to thank Jury, Gain, Navid, Behnoush, Jonathan, Can, Sebastian, Cosimo for always being ready to go out for a drink after work. My special thanks go to Kiarash who was my first choice whenever I needed an explanation of anything remotely related to the *analog world*, and to my office mate Şeniz who often had to suffer long and rather loud discussions. Two people who had the most important influence on my work were my Turkish brothers - Ömer and Kerem. Their help cannot be truthfully explained here, so I will just say - Thank you, guys, for everything. Finally, I absolutely must thank the Serbian people in Lausanne, especially Džoni, Ćojba, Mića, Velja and Kopta for spending hours and hours in heated discussions during our famous coffee breaks. I also want to thank Balać for still being one of my long-standing friends.
Finally, I would like to thank my parents for always being there for me, even when I was overloaded with work and not reachable for days. Thank you for doing everything possible to provide me a life that I am living right now.

Lausanne, December 2015 Vladan Popović

Abstract

The standard cameras are designed to truthfully mimic the human eye and the visual system. In recent years, commercially available cameras are becoming more complex, and offer higher image resolutions than ever before. However, the quality of conventional imaging methods is limited by several parameters, such as the pixel size, lens system, the diffraction limit, etc. The rapid technological advancements, increase in the available computing power, and introduction of Graphics Processing Units (GPU) and Field-Programmable-Gate-Arrays (FPGA) open new possibilities in the computer vision and computer graphics communities. The researchers are now focusing on utilizing the immense computational power offered on the modern processing platforms, to create imaging systems with novel or significantly enhanced capabilities compared to the standard ones. One popular type of the computational imaging systems offering new possibilities is a multi-camera system.

This thesis will focus on FPGA-based multi-camera systems that operate in real-time. The aim of the multi-camera systems presented in this thesis is to offer a wide field-of-view (FOV) video coverage at high frame rates. The wide FOV is achieved by constructing a panoramic image from the images acquired by the multi-camera system. Two new real-time computational imaging systems that provide new functionalities and better performance compared to conventional cameras are presented in this thesis. Each camera system design and implementation are analyzed in detail, built and tested in real-time conditions. *Panoptic* is a miniaturized low-cost multi-camera system that reconstructs a 360 degrees view in real-time. Since it is an easily portable system, it provides means to capture the complete surrounding light field in dynamic environment, such as when mounted on a vehicle or a flying drone. The second presented system, *GigaEye II* , is a modular high-resolution imaging system that introduces the concept of distributed image processing in the real-time camera systems. This thesis explains in detail how such concept can be efficiently used in real-time computational imaging systems.

The purpose of computational imaging systems in the form of multi-camera systems does not end with real-time panoramas. The application scope of these cameras is vast. They can be used in 3D cinematography, for broadcasting live events, or for immersive telepresence experience. The final chapter of this thesis presents three potential applications of these systems: object detection and tracking, high dynamic range (HDR) imaging, and observation of multiple regions of interest. Object detection and tracking, and observation of multiple regions of interest are extremely useful and desired capabilities of surveillance systems, in

security and defense industry, or in the fast-growing industry of autonomous vehicles. On the other hand, high dynamic range imaging is becoming a common option in the consumer market cameras, and the presented method allows instantaneous capture of HDR videos.

Finally, this thesis concludes with the discussion of the real-time multi-camera systems, their advantages, their limitations, and the future predictions.

Key words: Computational Imaging, FPGA, Real-time, Multi-camera, Gigapixel Imaging, Panorama, Omnidirectional

Résumé

Les caméras classiques sont conçues pour imiter fidèlement l'œil humain et le système visuel. Au cours des dernières années, les caméras disponibles sur le marché sont de plus en plus complexes, et offrent des résolutions d'images plus élevées que jamais. Cependant, la qualité des méthodes classiques de formation d'image est limitée par plusieurs paramètres tels que la taille du pixel, le système de lentilles, la limite de diffraction, etc. Les progrès technologiques, l'augmentation de la puissance de calcul disponible ainsi que l'introduction des unités de traitement graphique (GPU) et Field-Programmable-Gate-Arrays (FPGA) ouvrent de nouvelles possibilités dans les communautés de vision par ordinateur et d'infographie. Les chercheurs se concentrent maintenant sur l'utilisation de l'immense puissance de calcul offerte par les plates-formes modernes de traitement pour créer des systèmes d'imagerie avec des capacités nouvelles ou significativement améliorées par rapport à ceux existants. Un type populaire de systèmes d'imagerie de calcul offrant de telles possibilités nouvelles est un système multi-caméras.

Cette thèse se concentre sur les systèmes multi-caméras à base de FPGA fonctionnant en temps réel. L'objectif des systèmes multi-caméras présentés dans cette thèse est de proposer une couverture vidéo à large champ de vision (FOV) à des cadences élevées. Le large champ de vision est obtenu par la construction d'une image panoramique à partir des images acquises par le système multi-caméras. Deux nouveaux systèmes d'imagerie de calcul en temps réel offrant de nouvelles fonctionnalités et de meilleures performances par rapport aux caméras conventionnelles sont présentés dans cette thèse. Chaque conception et application du système de la caméra sont analysées en détail, construits et testés dans des conditions en temps réel. Panoptique est un système multi-caméras à faible coût miniaturisé qui reconstitue une vue à 360 degrés en temps réel. Etant un système facilement transportable, il fournit des moyens pour capturer le champ complet de la lumière ambiante dans un environnement dynamique, tel que lorsqu'il est monté sur un véhicule ou un drone volant. Le deuxième système présenté, GigaEye, est un système d'imagerie à haute résolution modulaire qui introduit la notion de traitement d'image distribué dans les systèmes de caméra en temps réel. Cette thèse explique en détail comment ce concept peut être utilisé efficacement dans des systèmes d'imagerie de calcul en temps réel.

Le but des systèmes d'imagerie de calcul sous forme de systèmes multi-caméras ne s'arrête pas qu'aux panoramas en temps réel. Le champ d'application de ces caméras est vaste. Ils

peuvent être utilisés dans le cinéma 3D, la diffusion d'événements en direct, ou pour avoir une expérience de télé-présence immersive. Le dernier chapitre de cette thèse présente trois applications potentielles de ces systèmes : la détection d'objet et de suivi, l'imagerie à grande gamme dynamique (HDR), et l'observation de plusieurs régions d'intérêt. La détection et le suivi d'objets ainsi que l'observation de plusieurs régions d'intérêt sont extrêmement utiles dans les systèmes de surveillance, l'industrie de sécurité et de défense ou dans l'industrie à croissance rapide de véhicules autonomes. D'autre part, l'imagerie à grande gamme dynamique est en train de devenir une option courante dans les caméras du marché grand public. La méthode présentée permet la capture instantanée de ce type de vidéos.

Enfin, cette thèse se termine par la discussion des systèmes temps-réel multi-caméras, leurs avantages, leurs limites et les prédictions futures.

Mots clefs : Imagerie Computationnelle, FPGA, Temps réel, Multi-caméras, Imagerie Gigapixels, Panorama, Omnidirectionnel

Contents

List of Figures

List of Tables

List of Acronyms

- **1D** - **One**-**D**imensional
- **2D** - **Two**-**D**imensional
- **3D** - **Three**-**D**imensional
- **ADC** - **A**nalog-to-**D**igital **C**onverter
- **AG** - **A**daptive **G**aussian
- **API** - **A**pplication **P**rogramming **I**nterface
- **ASIC** - **A**pplication-**S**pecific **I**ntegrated **C**ircuit
- **CCD** - **C**harged-**C**oupled **D**evices
- **CDS** - **C**orrelated **D**ouble **S**ampling
- **CIF** - **C**ommon **I**ntermediate **F**ormat
- **CMOS** - **C**omplementary **M**etal-**O**xyde-**S**emicondutor
- **COTS** - **C**ommercial **O**ff-**T**he-**S**helf
- **CPU** - **C**entral **P**rocessing **U**nit
- **CUDA** - **C**ompute **U**nified **D**evice **A**rchitecture
- **DDFS** - **D**irect **D**igital **F**requency **S**ynthesizer
- **DDR** - **D**ouble **D**ata **R**ate
- **DRAM** - **D**ynamic **R**andom **A**ccess **M**emory
- **DVI** - **D**igital **V**isual **I**nterface
- **FHE** - **F**orward **H**omography **E**stimator
- **FIFO** - **F**irst **I**n **F**irst **O**ut

- **FIR** - **F**inite **I**mpulse **R**esponse
- **FOV** - **F**ield-**o**f-**V**iew
- **FPGA** - **F**ield-**P**rogrammable-**G**ate-**A**rray
- **FPN** - **F**ixed-**P**attern **N**oise
- **fps** - **f**rames-**p**er-**s**econd
- **GP** - **G**aussian **P**yramid
- **GPGPU** - **G**eneral-**P**urpose **G**raphics **P**rocessing **U**nits
- **GPU** - **G**raphics **P**rocessing **U**nit
- **GUI** - **G**raphical **U**ser **I**nterface
- **HD** - **H**igh **D**efinition
- **HDMI** - **H**igh-**D**efinition **M**ultimedia **I**nterface
- **HDR** - **H**igh **D**ynamic **R**ange
- **HMD** - **H**ead-**M**ounted **D**isplay
- **HVS** - **H**uman **V**isual **S**ystem
- **I2C** - **I**nter-**I**ntegrated **C**ircuit
- **I/O** - **I**nput/**O**utput
- **LDR** - **L**ow **D**ynamic **R**ange
- **LP** - **L**aplacian **P**yramid
- **LUT** - **L**ook-**U**p **T**able
- **LVDS** - **L**ow-**V**oltage **D**ifferential **S**ignaling
- **MBB** - **M**ulti-**B**and **B**lending
- **NN** - **N**earest **N**eighbor
- **PC** - **P**ersonal **C**omputer
- **PCB** - **P**rinted **C**ircuit **B**oard
- **PRNU** - **P**hoto **R**esponse **N**on **U**niformity
- **PSNR** - **P**eak **S**ignal-to-**N**oise **R**atio
- **PTZ** - **P**an-**T**ilt-**Z**oom

- **PWL** - **P**iece-**W**ise **L**inear
- **RAM** - **R**andom **A**ccess **M**emory
- **RGB** - **R**ed-**G**reen-**B**lue
- **SCC** - **S**ine and **C**osine **C**alculator
- **SCIP** - **S**mart **C**amera **I**ntellectual **P**roperty
- **SoC** - **S**ystem-**o**n-**C**hip
- **SPI** - **S**erial **P**eripheral **I**nterface
- **SRAM** - **S**tatic **R**andom **A**ccess **M**emory
- **USB** - **U**niversal **S**erial **B**us
- **VESA** - **V**ideo **E**lectronics **S**tandards **A**ssociation
- **VGA** - **V**ideo **G**raphics **A**rray
- **XGA** - e**X**tended **G**raphics **A**rray
- **ZBT** - **Z**ero **B**us **T**urnaround
- **ZIF** - **Z**ero **I**nsertion **F**orce

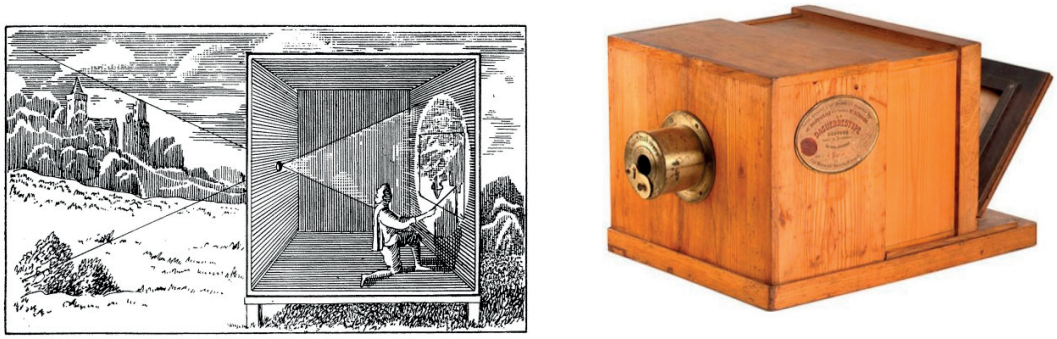
1 Introduction

For more than a century, photographs were taken by exploiting the chemical reaction of light rays hitting the photographic film. The photographic film is a plastic film that is coated with light-sensitive *silver-halide* crystals. The emulsion will gradually darken when exposed to light, but the process is too slow and incomplete to be of any practical use. A principle of *camera obscura* is used to resolve this problem. The *camera obscura* ("dark room" - Latin) is an optical device consisting of a box and a small hole (pinhole) on one of its sides, as shown in Figure 1.1a. Light that passes through the hole projects an inverted image of the world on the opposite side, but with preserved color and perspective.

Even though the first written records of *camera obscura* can be found in the 4^{th} century BC in the works of Chinese philosopher Mozi, and the Greek philosopher Aristotle, the first clear description and extensive experiments were published by Leonardo da Vinci in 1502. The modern cameras are based on this principle, but do not use the pinhole as it produces dim or blurry images, depending on the hole size. A lens, or a system of lenses, is used to focus the light, producing the usable brightness levels and sharp images. The image is projected on the photographic film placed on the opposite side of the lens. A short exposure to the image formed by a camera lens is used to produce only a very slight chemical change, proportional to the amount of light absorbed by each crystal. This creates an invisible latent image in the emulsion, which can be chemically developed into a visible photograph.

It was more than two hundred years until the first commercial camera was produced. The "Giroux Daguerreotype", in Figure 1.1b, was presented in Paris in 1839. It was considered to be a very fast camera, with the exposure time of around three minutes, depending on the scene illumination. The oldest surviving photograph is shown in Figure 1.2. It was shot with one of test prototypes of Giroux's camera.

Technological advancements in the late 20^{th} and early 21^{st} century also affected the camera manufacturing process. The cameras moved from the analog capture process using photographic film to the digital one. The photographic film has been replaced by the digital image sensors that convert light, *i.e.* photons, into electrical signals. Two main types of digital sensors



(a) (b)

Figure 1.1: (a) Illustrated principle of *camera obscura*, and (b) the first commercially available camera produced by Louis-Jacques Daguerre and Alphonse Giroux.

Figure 1.2: "View from the Window at Le Gras" is the oldest surviving photograph. It was taken in 1826 or 1827.

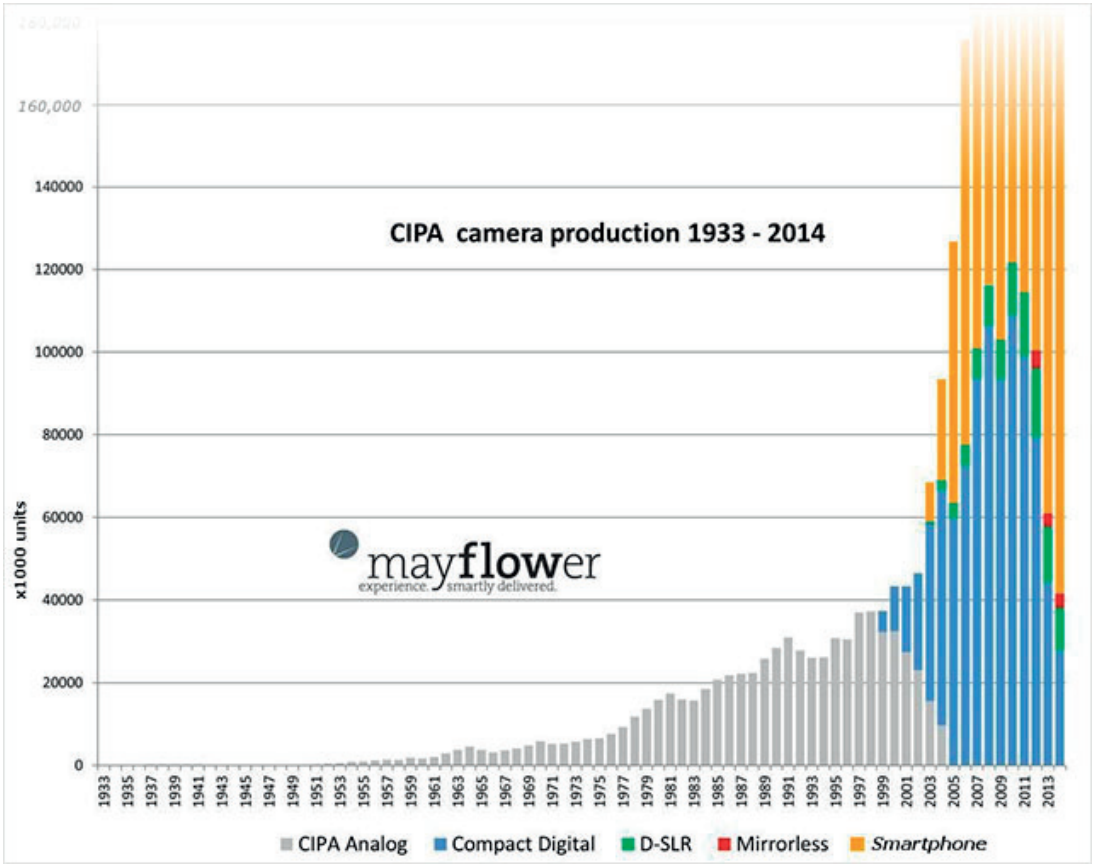


Figure 1.3: History of the camera production.

are used today: Charged-coupled devices (CCD) and active pixel sensor in complementary metal-oxyde-semiconductor (CMOS) technology. Most of the modern-day sensors use CMOS technology thanks to the lower price, simpler design, and faster readout of pixels. The big boom in the camera use started about a decade ago, with the appearance of mobile phones with the integrated cameras, and later smartphones, as shown in Figure 1.3. The cameras are now massively produced, they are easily accessible, and they are becoming cheaper.

1.1 Computational Imaging

Research in the fields of image processing and computer vision is based on obtaining truthful representation of the visual world, and adapting the acquired data to the desired purpose. The digital image sensors are used to measure the light intensities in the real world. Based on these measurement, various algorithms have been developed to reduce image acquisition artifacts (noise, lens distortion, chromatic aberration, etc.), compress the size of the image without losing quality, or apply some of the advanced methods for understanding contents of the image (edge detection, image segmentation, object recognition, etc.).

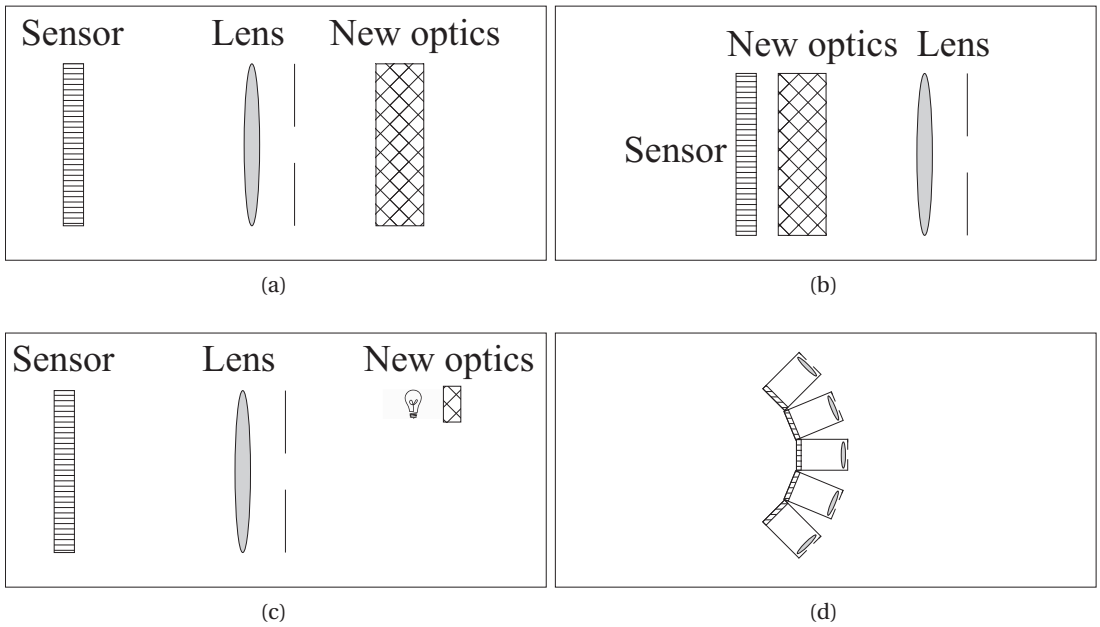


Figure 1.4: Various design methodologies of computational cameras [1]: (a) Object side coding, (b) focal plane coding, (c) light coding, and (d) multi-camera systems.

With the increased availability of the image sensors, the researchers started looking at the new possibilities in the image acquisition domain. Conventional imaging technique is limited to representing the world in a two-dimensional (2D) frame. The need to truthfully represent the environment initiated changes in the way the images are taken. Computational imaging, or computational photography, is a new vibrant research field in the last decade. It differs from the conventional imaging by utilizing a mix of novel optics and computational algorithms to capture the image. The novel optics is used to map rays in the light field of the scene to pixels on the detector in some unconventional fashion [1].

The fundamental idea of a computational camera is to use a combination of optics and sensors to optically encode scene information. The image captured by such a device does not necessarily look like the scene we see with our eyes. Hence, computational cameras have the processing (computing) step, which translates the recorded data into a meaningful image that our brain can understand. These systems are used to provide new functionalities to the imaging system (depth estimation, digital refocusing, super-resolution, etc.), or to increase system's performance (image resolution, camera frame rate, color reproduction, etc.).

Figure 1.4 illustrates the most popular design methodologies of computational cameras. Object side coding in Figure 1.4a is the easiest way to implement a computational camera. It denotes any kind of attachments external to the camera, such as adding a mirror (detailed in Section 2.3) or a prism. Figure 1.4b shows a focal plane coding method where an optical element is added close to the image sensor. The example of potential optical elements is a

Figure 1.5: The oldest surviving panoramic photograph. It shows the view of San Francisco in 1851 from Rincon Hill.

pixel filter for multispectral and high dynamic range imaging, or a microlens array used to capture the light field (Section 2.6). Figure 1.4c shows an example where the computational camera uses controlled illumination, such as the coded structured light [2]. Finally, Figure 1.4d shows an omnidirectional (*omni* - "all" in Latin) imaging system that utilizes a multiple camera setup. These systems will be the main topic of this thesis, starting from the design constraints and ending with several potential applications.

1.2 Bridging the Gap

Multi-camera systems with cameras arranged in a circle or a sphere are suitable for capturing panoramic images and videos. Panoramas were originally created by taking several photographs with a single camera, developing the film, and then manually stitching photographs one to another (Figure 1.5). It was mostly used in artistic, or military purposes, such as wide area surveillance. Today, there is a plethora of algorithms for the automatic image stitching and panorama construction. The most used ones will be explained in Chapter 3.

A common pipeline of panorama construction is to take images using one or more cameras, order them, and stitch into a single wide field-of-view (FOV) image. The stitching process is usually done offline on a personal computer (PC). Such approach does not have timing constraints, and the applied image processing algorithms can be very intensive and time-demanding. As a result, the image quality of the panorama can be very high.

However, real-time operation is needed in applications such as medical imaging for diagnostics or during surgeries, surveillance, or autonomous vehicles. Currently, the real-time operation is possible only with a single (or very few) camera and with a limited resolution. If a wide FOV is required, the systems are designed with a *fish-eye* lens, or with a camera mounted on a movable stand [3].

This thesis focuses on the design of computational imaging systems operating in real-time and with high image resolution. We use the high performance processing platforms based on field-programmable-gate-array (FPGA), which allow faster computations and real-time operation. FPGA systems are suitable for this purpose since they are programmable, and provide fast prototyping compared to the design of a dedicated application-specific-integrated-circuit (ASIC). A complete system-on-chip, which includes a microprocessor and its peripherals, can be inferred in the FPGA. In order to achieve the highest possible speed, the dedicated

hardware processing blocks are created. Hence, there is no image processing done on the microprocessor, resulting in significant increase in performance.

The currently existing panorama construction software algorithms cannot be efficiently ported to the hardware processing platforms. Hence, they need to be modified and adapted for fast processing on FPGA, without any loss of image quality compared to its software counterpart. Furthermore, we will present the design pipeline of a novel, modular and fully scalable multi-camera system, able to reach Gigapixel resolution, that still operates with a real-time frame rate of 30 frames-per-second (fps). Up to our knowledge, this is currently the fastest omnidirectional imaging system.

1.3 Key Contributions of the Thesis

Each result presented in this thesis has originated from and was motivated by a practical problem in the fields of computational imaging and embedded systems design. The problems at hand have defined the methods to be used in this work. Thus, the contributions of the thesis are presented by their corresponding applications, *i.e.* in each chapter we start by explaining the practical problem at hand, and then we present the methods to solve such a problem, together with their practical implementation.

The main contributions of this thesis can be classified in three categories: algorithm development for panorama construction, multi-camera system design, and novel camera applications. The contributions include for the first category:

- The panoramas are created by projecting the single camera images onto a surface, such as sphere or cylinder. The panoramas are then displayed by sampling the sphere using equiangular sampling. We provide **a novel constant pixel density sampling scheme**, which guarantees that each camera used in the system contributes to the panorama with approximately the same number of pixels. The advantage of this scheme is two-folded: the processing load is equalized among the cameras, and distortions in the spherical panoramas are reduced thanks to smaller influence of the cameras observing the poles.

- The formulation of the **novel image blending method named Gaussian blending**. This new method is based on a weighted average with the per-pixel-per-camera weights, as opposed to the standard per-camera weights. The Gaussian blending method resolves artifacts observed in the current state-of-the-art approaches, such as ghosting or visible seams. Furthermore, the Gaussian blending is **suitable for a fast real-time hardware implementation**.

The second category focuses on the hardware design of multi-camera systems, and the contributions include:

- The full design of the *Panoptic* camera that is **the first real-time omnidirectional multi-camera system**. It is also **the first computational imaging system to use an FPGA as the processing core**, which combined with the novel image processing architectures results in real-time **panoramic video** construction.

- The real-time hardware implementation of the **Forward Homography Estimator**. The proposed implementation results in the lower memory bandwidth and capacity requirements, and increased signal-to-noise ratio

- **The novel architecture of separable filters** for image decomposition into Laplacian pyramid. The presented implementation omits the image transpose operation, hence **avoiding storing the large intermediate results**. Furthermore, the introduced control logic **significantly reduces the power consumption** making the design suitable for any low-power application.

- **The fully modular and scalable multi-camera system**, GigaEye II, designed for **gigapixel video** construction. The system uses multi-layered processing architecture with stacked FPGA processing boards. This system is able to process gigapixels of data in real-time and create a panoramic video output at high resolutions and high frame rates. Compared to the state-of-the-art, GigaEye II is **the fastest multi-camera system**, and the whole processing pipeline is implemented in FPGA, hence it does not require a large cluster of computers for the image processing.

Finally, this thesis presents the new applications for the computational imaging systems:

- We present the first system targeted at **virtual reality applications** that includes both the acquisition device (*Panoptic* camera) and the virtual reality goggles as a display device (Oculus Rift).

- A new method to record and **reconstruct the high dynamic range video** using multiple cameras. We designed a multi-camera system able to utilize the overlap in the cameras' field-of-view to **recover the high dynamic range radiance map**, **stitch the panoramic frame**, and **apply the tone mapping** for proper display. All of the operations are done in real time.

- We introduce a **heterogeneous processing system** consisting of the multi-camera platform with FPGA processing, and the object detection implemented on a personal computer with Graphics Processing Unit (GPU). The processing load is divided such that operations suitable for pipelined processing are implemented on FPGA, whereas the ones suitable for parallel processing are implemented on GPU. This load distribution optimizes the total speed of the system for the given application.

1.4 Thesis Outline

This thesis is organized as follows. Chapter 2 gives an overview of state-of-the-art computational imaging systems and panorama construction algorithms.

Chapter 3 introduces the panorama construction algorithms, together with their advantages and disadvantages. The image formation in a single camera is also given, as well as the explanation of how we can use the camera geometry to correctly stitch the panorama. This chapter introduces illumination differences as the main problem in panorama generation. The major contributions of this chapter are the proposed Gaussian image blending solutions that can be implemented in a real-time camera system.

The second main contribution of this thesis is given in Chapter 4, where the first real-time multi-camera system is presented. It is a miniaturized and easily portable system. The chapter describes the full design procedure, including the image acquisition, real-time processing, and ways to immersively display the panoramic video.

The third main contribution of this thesis is given in Chapter 5, which presents the very high-resolution imaging system called *GigaEye II* . This chapter also explains all the considerations that have to be taken into account when designing such systems. Novel processing architectures and implementations are given in detail. Furthermore, the system is design in a modular fashion, and new cameras can easily be added to increase the resolution even further.

Finally, three applications of these computational imaging systems are given and demonstrated in Chapter 6. The three chosen applications are multi-view display, high dynamic range (HDR) imaging and object tracking.

Chapter 7 concludes this thesis and states ideas for the future research in this field.

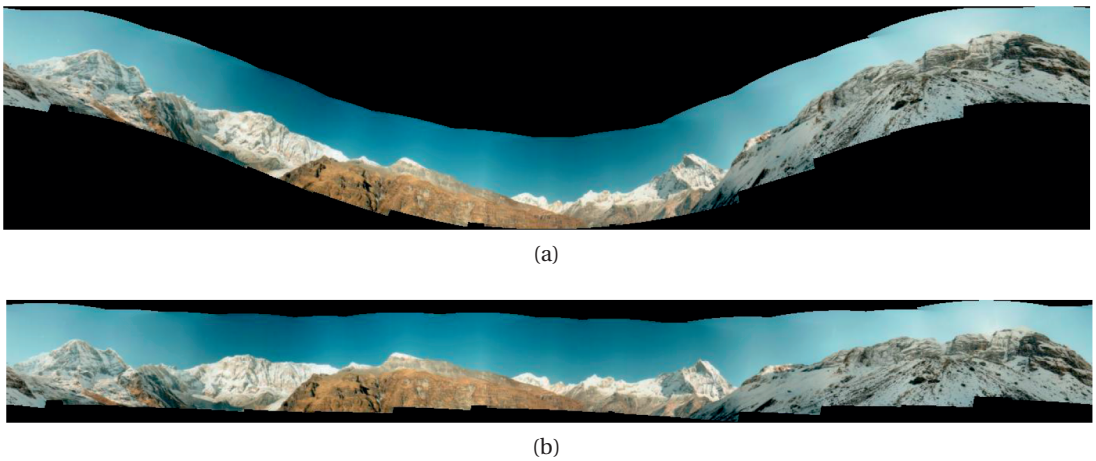
2 State-of-the-Art in Camera Systems

This chapter presents both pioneering and state-of-the-art algorithms and systems for acquisition of images suitable for wide FOV imaging. The most common systems include translational and rotational single camera systems, catadioptric cameras, multi-camera systems, and finally, commercially available standard and light-field cameras.

2.1 Panorama Stitching Algorithms

Panoramas have a much wider FOV than the standard cameras can provide. They are usually created by stitching several images taken by one or more cameras into a single one. The acquired images should have overlapping regions that are used for alignment of the original images. The creation of panoramas or image mosaics has been a popular research topic over the past years. The process of creating a panorama consists of various processing steps, which are named differently in the literature. For the sake of simplicity and consistency in this thesis, the process of stitching multiple images is divided into two main parts: (1) proper image alignment, and (2) seamless image blending.

The purpose of the image alignment is to determine the correct orientation and position of the original images in the final mosaic. Images taken by mobile devices are not horizontally aligned and often have slight pitch and roll rotations. Additionally, images taken by different cameras may have noticeable differences in scaling, due to varying fabrication process of the camera's optical system. Various algorithms for aligning the captured images were developed in the last two decades. Szeliski and Shum [4] presented a method to recover 3D rotations between the images. Brown and Lowe [5] use shift-invariant features to find matches between all images. Thanks to the matching between all images, this method is robust to image order, camera position, and scale. The algorithm also includes the automatic horizon detection that straightens the final image mosaic (shown in Figure 2.1). Another method for image alignment is to use a video stream of frames [6].



(a)

(b)

Figure 2.1: Panorama straightening introduced by Brown and Lowe [5]. The figure shows (a) automatically generated panorama without straightening, and (b) the same panorama after the horizon straightening.

Whereas image alignment determines the geometry of the image, the blending step handles the pixel intensity differences in the final mosaic. A major issue in creating photo-mosaics resides in the fact that the original images do not have identical brightness levels. This is usually caused by diverging camera orientations in space. Thus, some cameras acquire more light than the others. The problem manifests itself by the appearance of a visible seam in regions where the images overlap. The current algorithms can be divided in two categories: seam smoothing and optimal seam finding.

The seam smoothing algorithms reduce the color difference between two overlapped images to make the seam less noticeable and remove artifacts from the stitching process. The alpha blending [6, 7], *i.e.* blending based on a weighted average between pixels in every image, can reduce or even completely remove the seams. However, high frequency blurring may occur in the presence of any small image alignment error. If the alignment error occurs in the region where the objects can be found, it incurs the blending of a background pixel from one image with the foreground pixel from another. More advanced seam smoothing algorithms relate to the gradient domain image blending [8, 9, 10, 11, 12]. These algorithms use gradient domain operations to produce smoother color transitions and reduce color differences. Another possible solution to resolving color transition issue consists of using a multi-resolution blending algorithm [5, 13, 14] where high frequencies are combined in a small spatial range, thus avoiding blurring. Recently, a new approach has been proposed [15], where blending is performed using optical flow fields, providing high quality mosaics.

Optimal seam finding approaches [16, 17, 18] search for seams in overlapping areas along paths where differences between source images are minimal. The seams can be used to label each output image pixel with the input image that should contribute to it. The combination of optimal seam finding and seam smoothing for image stitching has also been used in panorama

(a) (b)

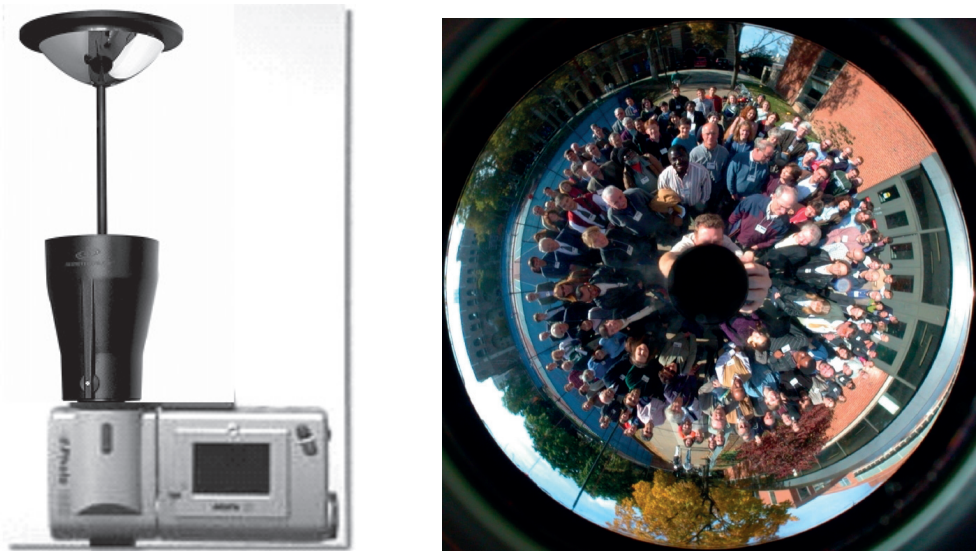
Figure 2.2: Two examples of single camera systems placed on a rotational stand. (a) Commercial PTZ camera Sony SNC-RZ30, and (b) BiCa360 camera [24].

applications [17]. Source images are first combined by compositing along optimal seams, and if the seams and stitching artifacts are visible, seam smoothing is applied to reduce color differences and hide the visible artifacts.

2.2 Single Camera Systems

Early systems for capturing multiple views were based on a translating or rotating high-resolution camera for capturing and later rendering the scene [19, 20, 21, 22]. They are sometimes called *slit cameras*, since they use only short strips of the scene to generate a panoramic image. A computer-controlled motorized mechanism rotates the camera holder in order to acquire the full 360° view. The advantage of the rotating camera is in its capability to acquire a high-resolution omnidirectional image, however at the cost of a long acquisition time. Therefore, it is difficult to use such systems to acquire a dynamic scene or a high frame rate video. Another disadvantage of these concepts is the limited vertical field-of-view, due to rotation around a single center. These problems were later resolved using a Pan-Tilt-Zoom (PTZ) camera [23], which is able to increase the vertical FOV by rotating around two axes.

Recently, the researchers started including special applications to the single camera systems. Belbachir et al. [25, 24] designed a BiCa360 smart camera that reconstructs a sparse panorama for machine vision applications.



(a) (b)

Figure 2.3: An example of catadioptric systems (a) with lens and a curved mirror, and (b) its image output.

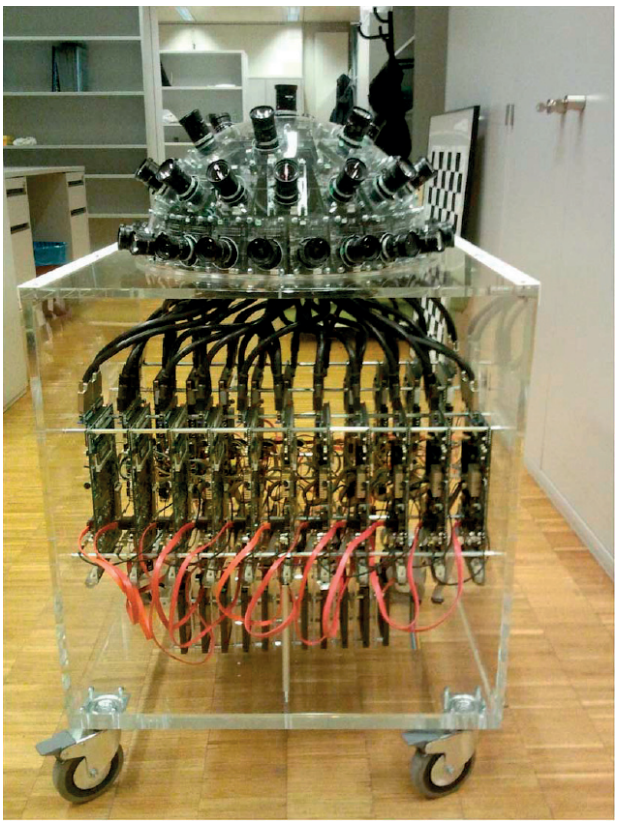
2.3 Catadioptric Systems

An alternate approach to omnidirectional acquisition is a catadioptric system [26], which consists of a convex mirror placed above a single camera sensor [27, 28]. The curved mirror of a catadioptric system is positioned in a way to reflect a 360° FOV environment directly into the lens. The mirror shape and lens are specifically chosen to allow a single viewpoint panorama, and easy unrolling of the acquired image to a cylindrical or spherical panorama.

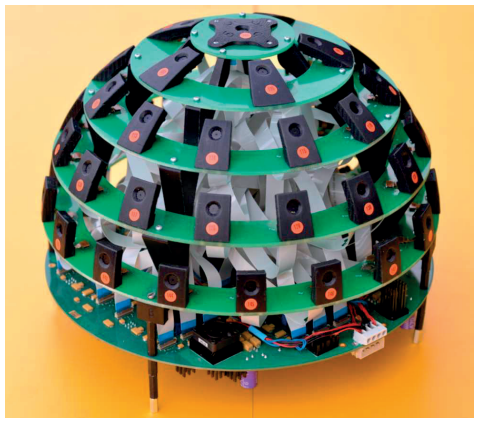
Catadioptric systems have the advantage of real-time and high frame rate video acquisition. Furthermore, the mirror is used to divert the light into the lens. Hence, there is almost no chromatic aberrations or distortions, which are seen in wide FOV images acquired using a *fish-eye* lens. However, catadioptric systems are limited to the resolution of the sensor. Furthermore, their overall FOV is limited, since it is restricted to the area below the sensor.

2.4 Polydioptric Systems

A polydioptric (*poly* - "in multiple ways", *dioptric* - "vision assistance by focusing light") camera is a camera that captures a multi-perspective subset of the light-ray space. In the scope of this thesis, discussion on polydioptric cameras will be restricted to multi-camera systems. The idea of using more than one camera has been first proposed by Taylor [29]. The use of linear array of still cameras allowed panoramic acquisition of dynamic scenes, which create ghosting effect in the rotating camera systems.



(a)



(b)

(c)

Figure 2.4: (a) Polydioptric system designed at EPFL [30] that is able to acquire 220 Mpixels; (b) Real-time polydioptric system from EPFL [31]; (c) The Stanford Multi-Camera Array [32].

Construction of panoramic videos requires large data sets, and researchers focused on developing systems with arrays of video cameras. The first camera array systems were built only for recording and later offline processing on PCs [33]. Other such systems [34, 35, 36, 37] were built with real-time processing capability for low resolution and low frame rates. A general-purpose camera array system was built by Wilburn et al. [32] with limited local processing at the camera level. This system was developed for recording large amounts of data and its intensive offline processing, and not for real-time operations. A similar system for creating high-resolution spherical panoramas was developed by Cogal et al. [30]. It was envisioned for a high-resolution surveillance of both ground and aerial vehicles. Finally, one of the most well-known polydioptric systems for large data collection is the Google Street view [38].

Recently, several systems were built for high-resolution acquisition with some sort of real-time processing. Schreer et al. [39] designed a multi-camera system for live video production, which includes multiple standard cameras, and omnidirectional camera, and PTZ cameras. Xu et al. [40] built a system with six cameras placed on ring structure, providing high definition output in a limited FOV. The central direction of the FOV can be changed in real-time using sliding mechanism.

2.5 Commercial Cameras

Some of the modern digital cameras have the panoramic mode as a built-in function. They include a special capturing hardware and firmware, usually bracketed capture and gyroscope measurement, as well as embedded image stiching. Even some low-end cameras include this capability, *e.g.* Pentax Optio RZ10. Different names are given to this option: Motion Panorama (Fuji), Sweep Panorama (Sony), Easy Panorama (Nikon), or Panorama (Olympus). These are all single-lens cameras which take shots in a burst and stitch them into a single still panoramic image, *i.e.* they do not have a panoramic video capability.

With the advancement of technology, more and more companies started investing in integration of cameras and dedicated image processing chips to create what is now called a "Smart camera". As of today, all of the previously mentioned panorama generation techniques reached the market, and they are still available for purchase. A Swiss company Seitz produces several models of Roundshot cameras [41], which is a single lens camera mounted on a rotating stand. Kogeto [42] sells a teleconference camera Jo, which is also based on a high-speed rotation of a single camera. Concerning multi-camera systems, the most popular ones are Pointgrey's Ladybug [43] (Ladybug5 is the latest one today) consisting of six CMOS image sensors, FullView [44] with a system including four cameras and flat mirrors, and Ricoh's THETA [45] that has two back-to-back sensors and fish-eye lenses.

FullView camera (Figure 2.5b) is especially interesting since its four cameras are pointed at four flat mirrors looking outward. Effectively, each camera is looking in a different direction, but from the same single viewpoint. Reflection off a planar mirror is always clear and sharp, irrespective of the size of the camera aperture and its position relative to the mirror. As a result, composite images are artifact-free and blur-free. The drawback of this method is its reliability on mirrors and its limited field of view in the vertical direction.

2.6 Light-field and Unconventional Cameras

One of the latest types of computational cameras fall in the group of light-field cameras, *i.e.* they capture the complete light field in the environment. This gives a user an opportunity to render multiple views, unlike with the classical camera. There are several approaches to capture the light field.

Yang et al. [46] and Pelican Imaging cameras [47] capture the light field with a planar array of cameras. Commercially available cameras, such as Lytro [48] and Raytrix [49], use a microlens array and the main lens. The array is square-shaped and placed in the position where the standard cameras have the sensor. The main lens focuses the image on the microlens grid that is placed just in front of the sensor. The obtained image resembles the image of the microlens grid from the backside, and the true image is reconstructed using computational algorithms. This method allows post-processing operations such as refocusing, or the viewpoint change. The main drawbacks are lower resolution and low speed, due to the needed processing.

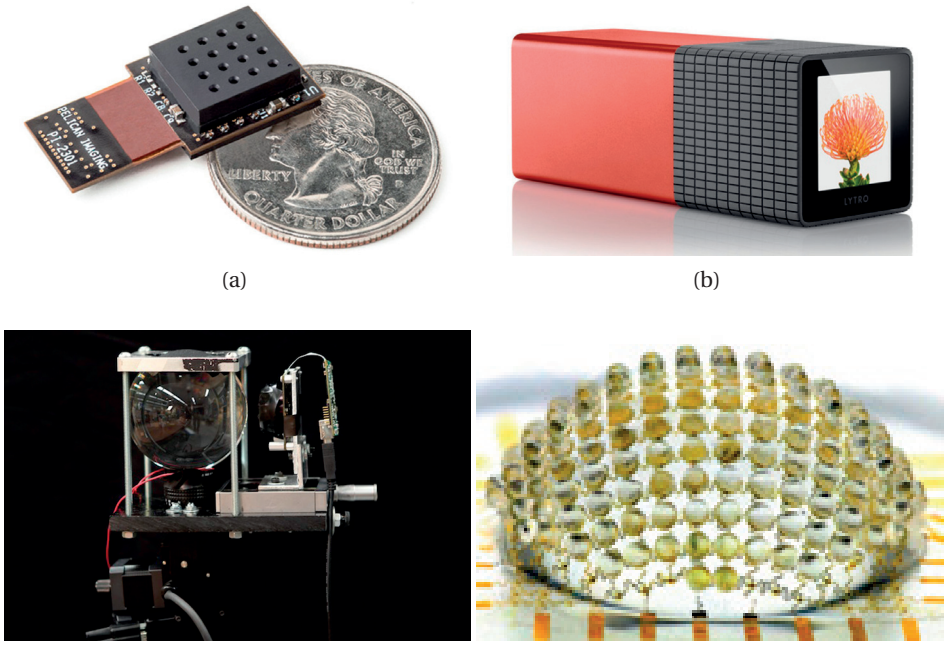
(a) (b)

(c) (d)

Figure 2.5: Commercially available panoramic cameras: (a) Pointgrey Ladybug5, (b) Working principle of FullView, (c) Seitz Roundshot D3, and (d) Ricoh THETA.

Recently, a camera system able to acquire an image frame with more than 1 gigapixel resolution was developed [50]. This camera uses a very complex lens system comprising of a parallel array of microcameras to acquire the image. Due to the extremely high resolution of the image, it suffers from a very low frame rate, even at low output resolution.

Some of the other interesting computational cameras include the work of Cossairt et al. [51] and Song et al. [52]. Cossairt et al. used multiple sensors arranged on a hemisphere looking inwards, and a single ball lens. This design also lacks the ability to process data with high frame rates. Song et al. designed a single camera in the shape of an arthropod eye, with the pixels manufactured on a stretchable surface. Due the manufacturing process and the technology limitations, this camera currently has only 256 pixels.



(a) (b)

(c) (d)

Figure 2.6: Light field cameras: (a) Pelican Imaging, (b) Lytro, (c) ball lens and a sensor array [51], and (d) artropod-inspired camera [52].

2.7 Conclusion

Solutions with multiple cameras offer higher and more uniform resolution than the single camera. Multiple cameras looking out directly into the scene from different positions see the world with inherently different perspectives. Hence each camera sees each point in space in a different direction. The difference in directions depends on the depth of the point of observation. This often makes it impossible to merge images into a single coherent image without image blending, which will be explained in Chapter 3. The image blending is a process that requires substantial image overlap and is highly prone to cause artifacts such as blurring or ghosting. Nevertheless, a multi-camera system with overlapping fields-of-view can be utilized for other target application at the same time or independently, *e.g.* extending the dynamic range, or object tracking.

Most developed camera-array systems are bulky and not easily portable platforms. Their control and operation depend on multi-computer setups. In addition to synchronization of the cameras, very large data rates present new challenges for the implementation of these systems. In the following section, a portable real-time multi-camera system will be presented in detail.

3 Panorama Construction Algorithms

In this chapter, we introduce panorama construction algorithms. These algorithms are developed for software processing on a PC, and their real-time implementation is often not possible. Thus, we provide modifications to the already existing algorithms, and develop a new one that can be implemented in hardware, and in real-time. The presented work results in improved image quality and the visually pleasant panorama for the human eye.

3.1 Fundamentals of Image Formation

The image formation can be approximated by the pinhole camera model for many computer vision applications [53, 54]. The pinhole camera projects a 3D world scene into a 2D image plane. In order to perform this projection, three coordinate systems are considered, as depicted in Figure 3.1. The coordinates of point $\mathbf{X} = [x\ y\ z]^T$ are expressed in the world coordinate system with its origin at the point $\mathbf{O}$. The same point can also be expressed in the camera coordinate system by coordinates $\mathbf{X_{cam}}$. Origin of the camera coordinate system coincides with the camera's focal point $\mathbf{O_c} = \mathbf{C}$. The relation between the world and camera coordinates is unique and consists of a single translation $\mathbf{t}$ and a single rotation R:

$$\mathbf{X_{cam}} = R(\mathbf{X} - \mathbf{t}) \tag{3.1}$$

The vector $\mathbf{t}$ denotes the distance between origins of the world and the camera coordinate systems, whereas the rotation matrix R expresses three Euler rotations in order to align the mentioned systems' axes. Parameters R and $\mathbf{t}$ are called extrinsic camera calibration parameters.

The axes of the image coordinate system are aligned with the camera coordinate system.

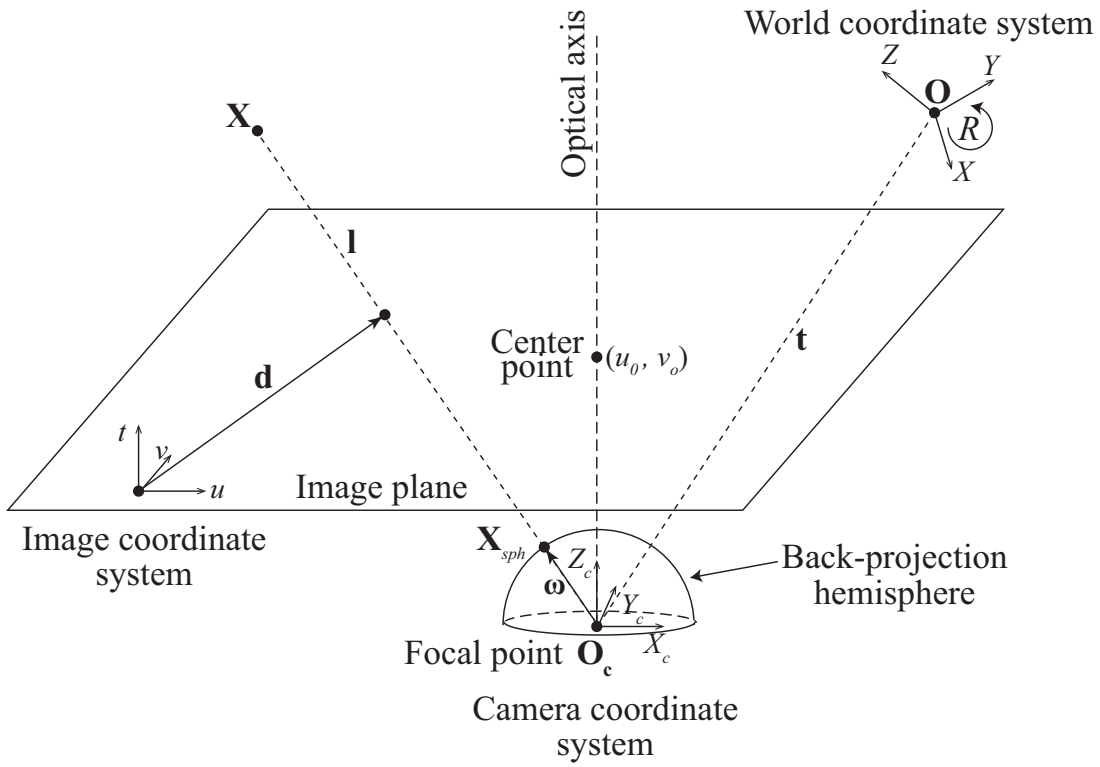
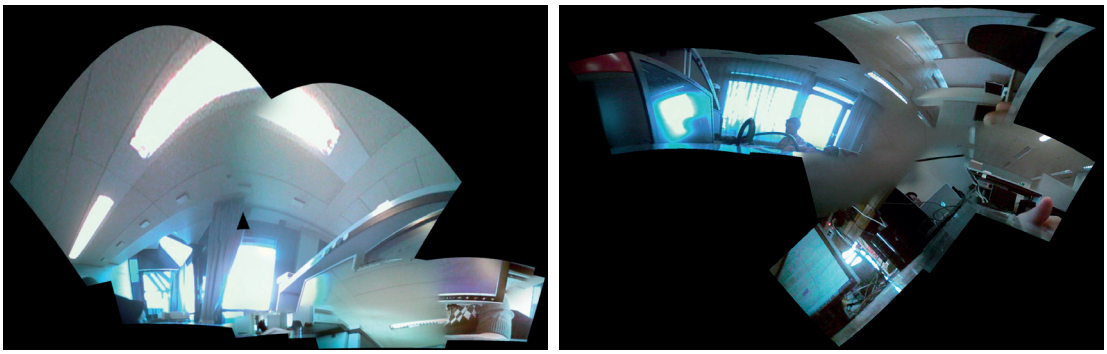


Figure 3.1: Geometric transformations during image formation process. The world, camera, and image coordinate systems are shown with relations between them. A light ray passing through the world scene point **X** and the sensor's focal point intersects the image plane in point **d**, which represents a pixel in the acquired image. The back-projection procedure reconstructs the original light ray **l** and locates the intersection point with the back-projection hemisphere, $\mathbf{X}_{sph}$.

Hence, position of the projected point in the image coordinate system is expressed by:

$$\mathbf{d} = M(\mathbf{X} - \mathbf{c}) = K\mathbf{X}_{\mathbf{cam}} = \begin{bmatrix} f & 0 & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X}_{\mathbf{cam}} \tag{3.2}$$

where **d** is the pixel position in the image coordinate system, f is the focal length, u_0 and v_0 are camera center (principal) point offsets, M is the projection matrix, and K is the intrinsic calibration matrix. The origin of the system is coincident with one of the corners of the image sensor. Thus, the shift of the sensor's central point (u_0, v_0) to the sensor's corner is needed to obtain the final pixel position.



(a) (b)

Figure 3.2: Incorrect panorama generation from fifteen input images, using the algorithm from [5]. The unexpected results are due to low number of feature points in the source images.

3.2 Image Stitching

Stitching of the panorama requires a correct alignment of all input images. In general, this can be done using uncalibrated system, *e.g.* mobile or handheld cameras, or using a stationary calibrated system. Most of the presented algorithm in Section 2.1 are automatic and work well in both system types. However, if the source image are noisy (low-light environment, low-quality sensor, etc.) or low resolution (specialized cameras, medical imaging), the automatic panorama generation often fails to correctly align and stitch the images. Figure 3.2 shows two examples of incorrect panorama generation using [5], due to insufficient number of feature points to make the proper matching. Both examples are panoramas generated from fifteen source images taken by a stationary calibrated system, presented in [55].

In this thesis, the focus is on fully calibrated systems and generating the panoramas as an omnidirectional view. For the sake of consistency, notation and the algorithms are explained for spherical view, *i.e.* projection of the source images onto the spherical surface, as shown in Figure 3.1. The omnidirectional vision of a virtual observer located anywhere inside the hemisphere can be reconstructed by combining the information collected by each camera in the light ray space domain (or light field [19]).

In this process, the omnidirectional view is estimated on a discretized spherical surface S_d of directions. The surface of this hemisphere is discretized into a grid with N_θ latitude and N_ϕ longitude samples, where each sample represents one pixel. The hemisphere S_d corresponds to the back-projection hemisphere from Figure 3.1. Figure 3.3a shows a discretized hemisphere with sixteen pixels for N_θ and N_ϕ each. A unit vector $\boldsymbol{\omega} \in S_d$, represented in the spherical coordinate system $\omega = (\theta_\omega, \phi_\omega)$, is assigned to the position of each pixel $\mathbf{X}_{sph}$. Different pixel distributions over the hemisphere are discussed in Section 3.2.1.

The construction of the virtual omnidirectional view $\mathscr{L}(\mathbf{q}, \boldsymbol{\omega}) \in \mathbb{R}$, where $\mathbf{q}$ determines the location of the observer, consists of finding all light rays $\mathbf{l}$, and their respective projections

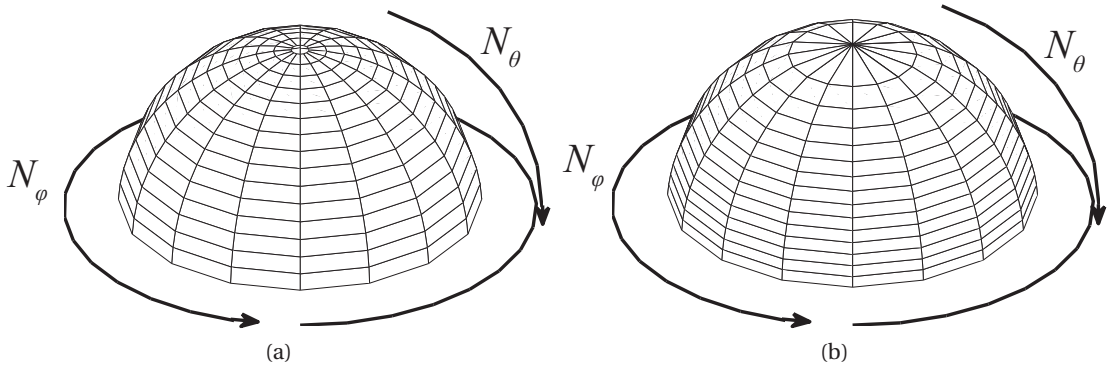


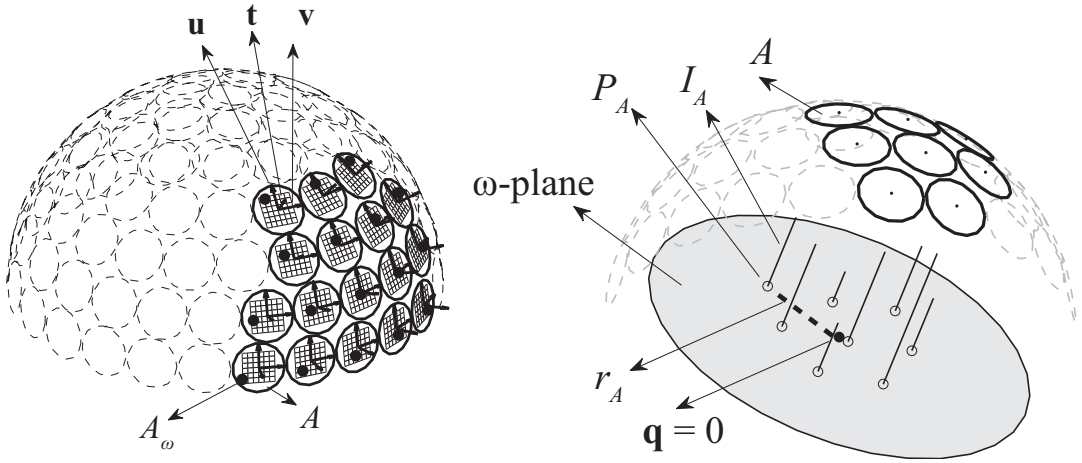
Figure 3.3: Pixelized hemispherical surfaces S_d with $N_\theta = 16$ latitude pixels and $N_\phi = 16$ longitude pixels (total of 256 pixels) using (a) equiangular and (b) constant pixel density discretization.

onto S_d. This approach requires three processing steps. The first step is to find all the cameras that capture the light ray **l**. The second step consists of finding a pixel in each input image that corresponds to the direction defined by $\boldsymbol{\omega}$. Finally, the third step consists of blending all pixel values corresponding to the same $\boldsymbol{\omega}$ into one. The result is the reconstructed light ray intensity $\mathscr{L}(\mathbf{q}, \boldsymbol{\omega})$.

To reconstruct a pixel in the omnidirectional view, all images having the observed $\boldsymbol{\omega}$ in their FOV are selected. Calibration of the system provides both intrinsic and extrinsic parameters [56]: focal length, center point offset, **t**, **u**, **v**, and angle-of-view α. The $\boldsymbol{\omega}$ is within the camera's FOV if the following constraint is met:

$$\boldsymbol{\omega} \cdot \mathbf{t_i} > \cos\frac{\alpha}{2}, \quad i = 1 \,..\, N_{cam} \tag{3.3}$$

Figure 3.4a illustrates a hemispherical camera arrangement, where camera positions are marked with circles. The full circles represent the cameras that have the observed $\boldsymbol{\omega}$ in their FOV. To extract the light intensity in that direction for each contributing camera, a pixel in the camera image frame has to be found. Due to the rectangular sampling grid of the cameras, the $\boldsymbol{\omega}$ does not coincide with the exact pixel locations on the camera image frames. Observing Figure 3.1, we need to find the closest pixel position **d** to the light ray **l**, *i.e.* ω. The exact position $\mathbf{d}' = [x, y]^T$ is obtained by finding the intersection point of **l** and the image frame.



(a) (b)

Figure 3.4: (a) Cameras contributing to the direction $\boldsymbol{\omega}$ with their contributing pixels in the respective image frames, (b) projections of camera centers contributing in direction $\boldsymbol{\omega}$ onto planar surface perpendicular to $\boldsymbol{\omega}$. P_A represents the projected focal point of camera A and I_A represents the pixel intensity. r_A is the distance of the projection point of the camera center from the virtual observer $\mathbf{q}$.

Using the pinhole camera model [54] and the assumption of a unit sphere:

$$\begin{aligned} \| \boldsymbol{\omega} \|_2 &= 1 \\ \mathbf{d}' &= (M\ \mathbf{l})^T \\ \mathbf{d} &= round(\mathbf{d}') \end{aligned} \tag{3.4}$$

In Descartes world coordinates, this corresponds to:

$$(d'_x, d'_y) = -(\boldsymbol{\omega} \cdot \mathbf{u}, \boldsymbol{\omega} \cdot \mathbf{v}) \frac{f}{\boldsymbol{\omega} \cdot \mathbf{t}} \tag{3.5}$$

The pixel location is chosen using the nearest neighbor method, where the pixel closest to the desired direction is chosen as an estimate of the light ray intensity. The process is then repeated for all $\boldsymbol{\omega}$ and results in the pixel values $I(c_i, \omega)$, where c_i is the radial vector directing to the center position of the i^{th} contributing camera. Figure 3.4a shows an example of the contributing cameras for an arbitrary pixel direction $\boldsymbol{\omega}$. The contributing position A_ω of the camera A, providing $I(c_A, \omega)$ is also indicated in Figure 3.4a.

The third reconstruction step is performed in the space of light rays given by direction $\boldsymbol{\omega}$ and passing through the camera center positions. Under the assumption of Constant Light Flux (CLF), the light intensity remains constant on the trajectory of any light ray. Following the CLF assumption, the light ray intensity for a given direction $\boldsymbol{\omega}$ only varies in its respective orthographic plane. The orthographic plane is a plane normal to $\boldsymbol{\omega}$. Such plane is indicated as the "ω-plane" in Figure 3.4b, and represented as a gray-shaded circle (the boundary of the circle is drawn for clarity purposes). The light ray in direction $\boldsymbol{\omega}$ recorded by each contributing camera intersects the ω-plane in points that are the projections of the cameras focal points on this plane. The projected focal points of the contributing cameras in $\boldsymbol{\omega}$ direction onto the ω-plane are highlighted by hollow points in Figure 3.4b. Each projected camera point P_i on the planar surface is assigned the intensity value $I(c_i, \boldsymbol{\omega})$.

As an example, the projected focal point of camera A onto the ω-plane (*i.e.* P_A) in Figure 3.4b is assigned the intensity value I_A. The virtual observer point inside the hemisphere (*i.e.* $\mathbf{q}$) is also projected onto the ω-plane. The light intensity value at the projected observer point (*i.e.* $\mathscr{L}(\mathbf{q}, \boldsymbol{\omega})$) is estimated by one of the blending algorithms, taking into account all $I(c_i, \boldsymbol{\omega})$ values or only a subset of them. In the given example, each of the seven contributing cameras shown with bold perimeter in Figure 3.4b provides an intensity value which is observed in direction $\boldsymbol{\omega}$ for the observer position $\mathbf{q} = 0$. The observer is located in the center of the sphere and indicated by a bold dot. A single intensity value is calculated from the contributing intensities through a blending procedure on its respective ω-plane. The process is repeated for all $\boldsymbol{\omega}$ directions to create a full 360° panorama.

3.2.1 Sphere Discretization

The pixel arrangement $\boldsymbol{\omega}$ shown in Figure 3.3a is derived from an equiangular distribution of the longitude and the latitude coordinates of a unit sphere into N_ϕ and N_θ segments, respectively. The equiangular distribution is defined by equal longitude and latitude angles between two neighboring pixels. This discretization enables the rectangular presentation of the reconstructed image suitable for standard displays, but results in the unequal contribution of the cameras. The density of the pixel directions close to the poles of the sphere is higher compared to the equator of the sphere in the equiangular scheme. Hence, the cameras positioned closer to the poles of the sphere contribute to more pixels in comparison to the other cameras in the system. The equiangular distribution is derived mathematically from (3.6):

$$\begin{aligned} \phi_\omega(i) &= \frac{2\pi}{N_\phi} \times i, \quad 0 \le i < N_\phi \\ \theta_\omega(j) &= \frac{\pi}{2N_\theta} \times (j + \frac{1}{2}), \quad 0 \le j < N_\theta \end{aligned} \tag{3.6}$$

Spherical panoramas often have more details around the equator than on the poles. Furthermore, the systems with camera arrangement as in Figure 3.4a place more cameras in the bottom rows, which allows higher spatial resolution. A constant pixel density scheme results in an approximately even contribution of the cameras. The scheme is based on enforcing a constant number of pixels per area, as expressed in (3.7). Compared to the equiangular distribution, the difference is observed in latitude angles. The discretization scheme expressed in (3.8) is derived by solving the integral in (3.7). The illustration of the constant pixel density sphere is shown in Figure 3.3b.

$$\frac{N_\phi \times j}{\int_0^{2\pi} d\phi \int_0^{\theta_\omega(j)} \sin\theta \, d\theta} = \frac{N_\phi \times N_\theta}{2\pi}, \quad 0 \le j < N_\theta \tag{3.7}$$

$$\begin{aligned} \phi_\omega(i) &= \frac{2\pi}{N_\phi} \times i, \quad 0 \le i < N_\phi \\ \theta_\omega(j) &= \arccos(1 - \frac{j}{N_\theta}) + \theta_0, \quad 0 \le j < N_\theta. \end{aligned} \tag{3.8}$$

The offset value θ_0 is added to the latitude angle in (3.8) to avoid repetition of pixel direction for the $j = 0$ case.

Figure 3.5a and Figure 3.5b illustrate the differences between two discretization methods in panorama construction, and Figure 3.5c and Figure 3.5d show the per-camera pixel contribution. Camera contribution in the constant pixel density distribution is almost equal for all cameras, whereas the top camera (looking in the north pole direction) in equiangular distribution contributes three times more than any other camera.

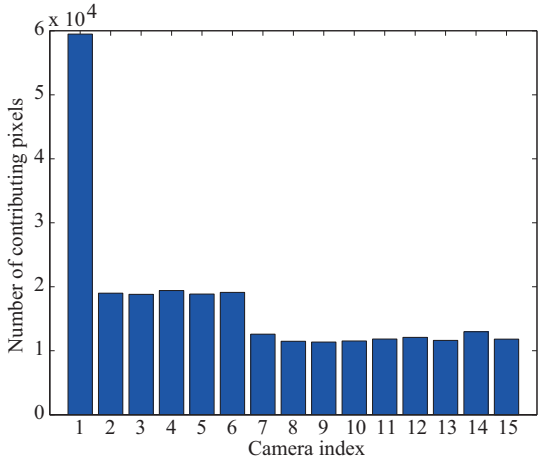
Latitudal pixel distribution does not need to be a linear or a trigonometrical function. Moreover, it can be any function, including any of the Piece-Wise Linear (PWL) ones. PWL functions are of special interest, since the pixel emphasis can be placed on several places on the sphere. To achieve such discretization, the full latitudal FOV of $\pi/2$ is divided into M pieces of arbitrary FOV_i, where each piece is linearly sampled. A number of pixels p_i is chosen for each of the pieces, separately, based on the desired application and view specifications. The latitude angles in each segment are linearly generated with an angular slope expressed in (3.9):

$$\Delta\theta_i = \frac{FOV_i}{p_i}, \quad 1 < i \le M \tag{3.9}$$

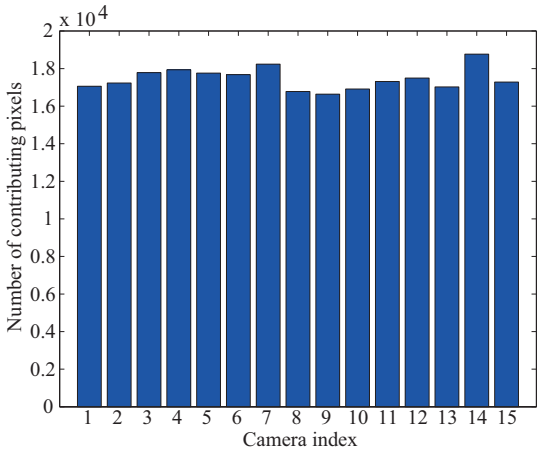
A comparison between the presented pixel distribution schemes is shown in Figure 3.6. An arbitrary PWL function comprising $M = 3$ pieces is taken for illustration purposes. This function results in higher pixel density near the pole and around the equator. The constant

(a)

(b)



(c)



(d)

Figure 3.5: A computer laboratory at the Swiss Federal Institute of Technology in Lausanne (EPFL, ELD227). Panoramic construction with a pixel resolution of $N_\phi \times N_\theta = 1024 \times 256$ (a) using the equiangular discretization, (b) using constant pixel density discretization. Figures (c) and (d) show the number of pixels each camera contributes to in case of (c) equiangular, and (d) constant pixel density discretization.

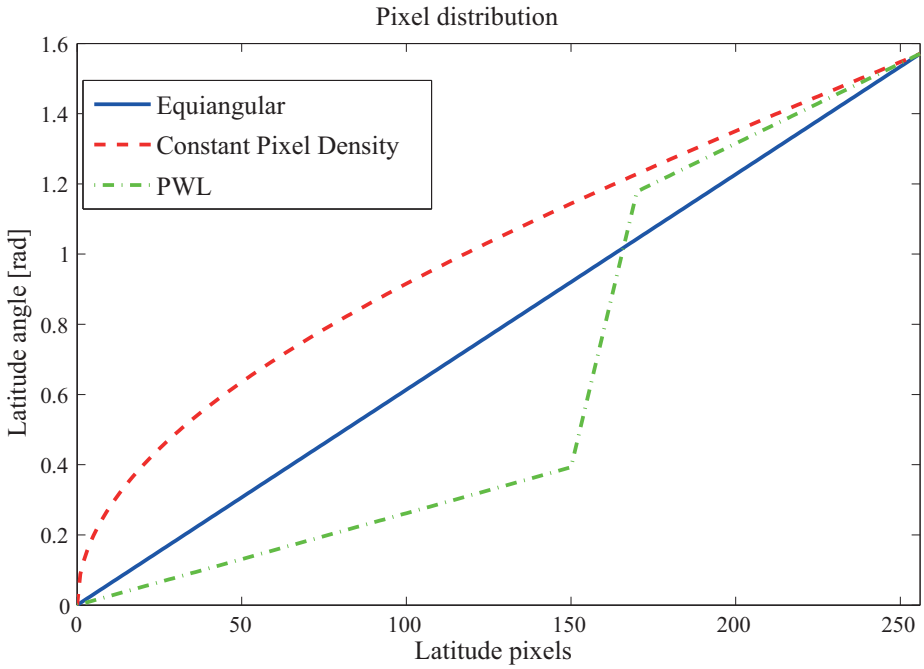


Figure 3.6: Latitude angle distribution for $N_\theta = 256$ latitude pixels using three different pixel distribution schemes.

pixel density scheme provides more pixels around the equator, *i.e.* when latitude angles are higher. Finally, the equiangular distribution provides linearly distributed pixels around the hemisphere.

Apart from region selectivity, the PWL scheme is used for approximation of functions such as logarithms or exponentials. The PWL approximation of such functions is necessary for a compact hardware implementation. These functions can be used when more details are required around the pole or the equator, respectively.

3.2.2 Grid Refinement

The presented discretization schemes can be regarded as sampling grids of the surrounding light field. The total number of acquired pixels linearly increases with the number of cameras. Thus, the light field can become oversampled using only several low-resolution cameras. Light field information is obtained at the subpixel scale as a benefit of this particular light field oversampling. Hence, the acquired images contain finer detail than shown on a fixed-grid rendered panorama. In addition to the fact that the resolution of the reconstructed image can be significantly smaller than the total number of acquired pixels, this creates an excess of pixels that are not used in the reconstruction process.

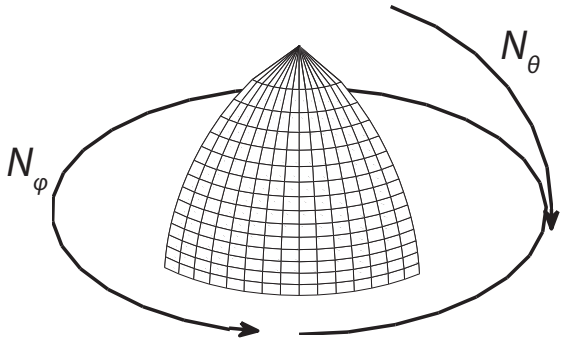


Figure 3.7: Refined pixelization scheme with $N_\theta = 16$ latitude pixels and $N_\phi = 16$ longitude pixels. Longitudal FOV is reduced to a quarter of the hemisphere.

Figure 3.8: Detailed image parts obtained using Gaussian blending with the grid refinement: a lamp magnified 8x, the books magnified 32x, a desk magnified 8x.

Nevertheless, the acquisition of the excess pixels can be useful. If an $\boldsymbol{\omega}$ direction in the reconstructed image is observed by more than one camera, *i.e.* parallax[1] exists in each point in space, a subpixel resolution can be achieved.

As presented in Section 3.2.1, it is possible to change the pixel distribution schemes. Additionally, the desired FOV is also programmable. Hence, a constant output resolution with the reduced FOV results in a grid refinement effect. The example of the refined pixelization is shown in Figure 3.7, where the increase in pixel density can be noticed in the desired FOV.

The effect observed in the reconstructed image is similar to the effect of digital zoom. However, the subpixel data is taken from the real and previously unused measured data, *i.e.* it is not calculated using an interpolation function as in digital zooming. Hence, grid refinement provides a more truthful light field rendering than digital zoom, and it is shown in Figure 3.8.

[1] Parallax is a displacement or difference in the apparent position of an object viewed along two different lines of sight

Figure 3.9: The image on left shows the vignetting effect as dark regions around the edges and corners of the image. The image on the right is the same image after the correction is applied. [Online source: http://intothenightphoto.blogspot.ch]

3.3 Vignetting Correction

Vignetting is an adverse effect observed in cameras, where the pixels located close to the image frame borders are significantly darker than the pixels located in the center. Vignetting also affects the reconstructed omnidirectional view; thus, pixel intensities in the reconstructed image alternatively vary, *i.e.* certain regions are darker and others are brighter. An example of vignetting effect is shown in Figure 3.9, where the left and the right image represent images before and after correction, respectively.

Several methods are proposed in literature for modeling the vignetting effect and its correction. The chosen model for Panoptic camera is the Kang–Weiss model [57]. The Kang–Weiss model takes into account the pixel position in the camera image frame, the camera focal length and a camera constant named the vignetting factor. All pixels in each camera frame are corrected by multiplying the sampled pixel intensity with a correction factor. The corrected pixel intensity is expressed as:

$$I'(u,v) = I(u,v)(1-\alpha d)\frac{1}{(1+(d/f)^2)^2} \tag{3.10}$$

where α is the vignetting factor, f is the focal length, $I(u,v)$ is the original pixel intensity at coordinates (u,v) and $d = \sqrt{u^2+v^2}$.

3.4 Alpha Blending

The first step in omnidirectional vision construction discussed in Section 3.2 consists of determining contributing pixels from input image frames and their respective intensities, $\mathscr{L}(c_i, \boldsymbol{\omega})$. Ideally, intensity values corresponding to the same direction will be equal in all the images. However, various parasitic effects, such as fabrication process, lens distortion, or exposure difference, influence difference in intensity. Furthermore, the stitching algorithm detailed in the previous section assumes that all the objects are far away from the camera. In reality, this is not always true, and the pixel intensities may differ significantly. Also, the obtained values may significantly vary due to diverging camera orientations, difference in light incidence angle, and misalignment of pixels due the calibration procedure. Even though the vignetting correction equalizes brightness of the individual camera's image, the reconstructed image quality mostly depends on the blending algorithm.

Alpha blending is a simple blending process where pixel intensities $\mathscr{L}(c_i, \boldsymbol{\omega})$ from all contributing cameras are weight-averaged. Different blending effects are obtained by changing the weights.

A special case of the alpha blending is the nearest neighbor (NN) blending. When applying the NN blending, the light intensity at the virtual observer point for each $\boldsymbol{\omega}$ direction is set to the light intensity value of the best observing camera for that direction. In terms of weighted average, this corresponds to the case when the weight of only one camera is 1, while the other cameras contribute with a 0 weight. The NN blending is expressed in (3.11) in mathematical terms:

$$\begin{aligned} &\mathscr{L}(\mathbf{q}, \boldsymbol{\omega}) = \mathscr{L}(c_j, \boldsymbol{\omega}) \\ &j = \text{argmin}_{i \in I}(r_i) \end{aligned} \tag{3.11}$$

where $I = \{i | \boldsymbol{\omega} \cdot \mathbf{t_i} \geq \cos(\frac{\alpha_i}{2})\}$ is the index of the subset of contributing cameras for the pixel direction $\boldsymbol{\omega}$. A pixel direction $\boldsymbol{\omega}$ is assumed observable by the camera c_i if the angle between its focal plane vector $\mathbf{t_i}$ (see Figure 3.4a) and the pixel direction $\boldsymbol{\omega}$ is smaller than half of the angle of view α_i of camera c_i. The length r_i identifies the distance between the projected focal point of camera c_i and the projected virtual observer point on the ω-plane. The camera with the smallest r distance to the projected point of virtual observer on the ω-plane is considered the best observing camera. As an illustration, such distance is identified with r_A and depicted by a dashed line for the contributing camera A in Figure 3.4b.

The NN blending does not resolve intensity differences between pixels from different images, and results in clearly visible seams [6, 55, 58]. The resulting image examples can be seen in Figure 3.10a. The artifacts caused by different brightness levels between cameras and misalignment can be resolved to a certain extent using linearly distributed weights.

(a)

(b)

(c)

(d)

Figure 3.10: A computer laboratory at the Swiss Federal Institute of Technology in Lausanne (EPFL, ELD227). Panoramic construction with a pixel resolution of $N_\phi \times N_\theta = 1024 \times 256$ (a) using the nearest neighbor technique, (b) using alpha blending, (c) using Gaussian blending with $\sigma_d = 100$ and (d) using Adaptive Gaussian blending with $\sigma_d = 100$ and $\sigma_r = 1/30$.

The linear weighting incorporates all the cameras contributing into a selected $\boldsymbol{\omega}$ direction through a linear combination [58, 59]. This is conducted by aggregating the weighted intensities of the contributing cameras. The weight of a contributing camera is the reciprocal of the distance between its projected focal point and the projected virtual observer point on the ω-plane, *i.e.* r_A in Figure 3.4b. The weights are also normalized to the sum of the inverse of all the contributing camera's distances. It is mathematically expressed in (3.12):

$$\begin{aligned} \mathscr{L}(\mathbf{q},\boldsymbol{\omega}) &= \frac{\sum\limits_{i\in I} w_i \cdot \mathscr{L}(c_i,\boldsymbol{\omega})}{\sum\limits_{i\in I} w_i} \\ w_i &= \frac{1}{r_i} \end{aligned} \tag{3.12}$$

3.5 Gaussian Blending

The NN and linear weighting present several issues. An image reconstructed using the NN method shows clear boundaries between the fields of view of different cameras. Although some brightness differences are reduced by the vignetting correction, the boundaries are still visible and create an unpleasant effect to the human eye.

Linear weighting solves the problem of sharp boundaries to a certain extent. Pixels in the regions where cameras' fields of view overlap are blended using a weighted average, as expressed in (3.12). The intensity difference is reduced, but it is still existant. Moreover, the main disadvantage lies in the appearance of blurred edges in the image due to the misalignment and linearly chosen weights.

We propose a two-fold modification of the alpha blending algorithm. The first modification relates to weights being a function of not only the camera's physical orientation $\mathbf{t}$, but also of the camera's intrinsic parameters and the position of the virtual observer. This modification is realized by adding a multiplicative factor to the linear alpha blending that is dependant on the pixel position d within the frame.

The second modification concerns the calculation of the multiplicative factor. By conducting subjective tests, it was empirically deduced that distributing the weights using a Gaussian function with respect to the pixel distance from the frame center results in the best image mosaic quality [55]. The image quality is improved in terms of both reducing visibility of the seams, and reducing the ghosts around edges of the scene objects. The new weights in the weighted average expression are:

$$\begin{aligned} w_{i,j} &= \frac{1}{r_i} \cdot \mathscr{G}(d_j,\sigma_d) \\ \mathscr{G}(d_j,\sigma_d) &= e^{-\frac{d_j^2}{2\sigma_d^2}} \end{aligned} \tag{3.13}$$

where r_i is the same distance as in (3.12), d_j is the distance of the j^{th} pixel in the input image frame from its center, and σ_d is the variance of the Gaussian distribution function $\mathscr{G}$.

By adding the Gaussian factor to the weighted average expression, the seams between cameras' FOVs are not visible any more, as shown in Figure 3.10c. Furthermore, the Gaussian blending reduces the difference in brightness in the images from different cameras and the overlapping regions are equalized with their respective surroundings. High-frequency blur is also reduced compared to the linear alpha blending weights.

3.5.1 Adaptive Gaussian Blending

The NN blending proves to be suitable for processing the pixels which are close to the camera center. Towards the boundaries of the camera's FOV, Gaussian blending is favorable thanks to the brightness equalization and reduction of effects originating from the camera misalignment. The Adaptive Gaussian (AG) blending technique aims to restrict the Gaussian blending to the areas where the reconstructed pixels are not close to the center of a single camera's FOV. The NN blending is used in the areas close to the mentioned centers. Hence, this method benefits from the advantages of both Gaussian and NN blending.

One way of implementing this method consists of simultaneously constructing the two views and blending them for the output display. However, the implementation of such approach is computationally demanding, and its real-time implementation is not possible with the current technology.

An efficient implementation of the AG blending is proposed. A new confidence factor is introduced, which is related to each camera's observation of a given $\boldsymbol{\omega}$ direction. For that purpose, a dot product of the $\boldsymbol{\omega}$ and the focal vector $\mathbf{t}$ (see Figure 3.4a) is taken as a reference metric.

In the blending phase of the reconstruction, a Gaussian confidence factor with respect to its $\boldsymbol{\omega} \cdot \mathbf{t_i}$ is multiplied with the previously calculated $w_{i,j}$ of each camera c_i obtained from the Gaussian blending technique. By expanding (3.13), the AG blending weight and the Gaussian confidence factor are expressed in mathematical terms:

$$\begin{aligned} \tilde{w}_{i,j} &= \frac{1}{r_i} \cdot \mathscr{G}(d_j, \sigma_d) \cdot \mathscr{C}(\boldsymbol{\omega}, c_i) \\ \mathscr{C}(\boldsymbol{\omega}, c_i) &= e^{-\frac{(\boldsymbol{\omega} \cdot \mathbf{t_i} - 1)^2}{\sigma_r^2}} \end{aligned} \tag{3.14}$$

where $\tilde{w}_{i,j}$ represents the new blending weight for j^{th} pixel in the i^{th} camera frame and $\mathscr{C}$ represents the AG confidence factor.

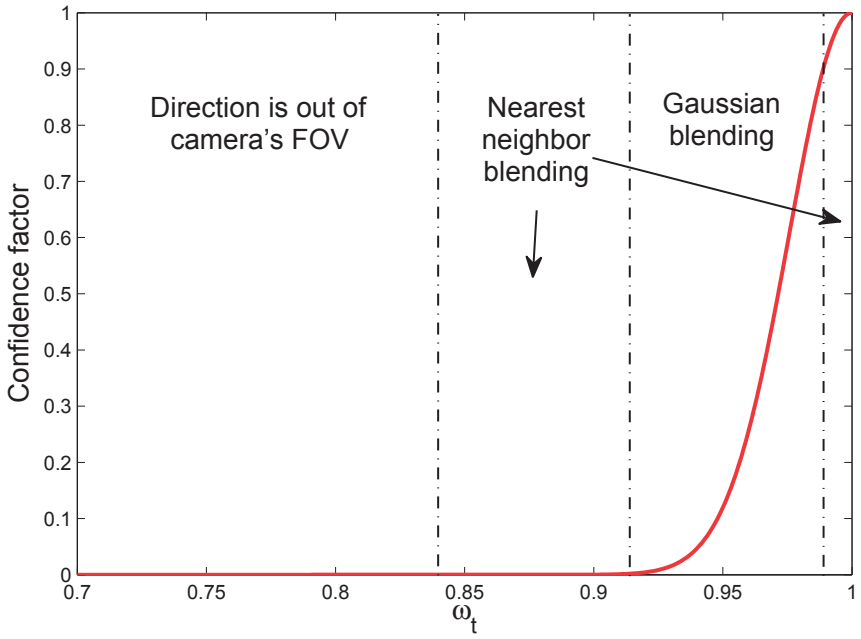


Figure 3.11: Confidence factor based on $\omega_t = \boldsymbol{\omega} \cdot \mathbf{t}$. Gaussian blending is applied only in the region where the confidence factor is lower than 0.9.

The AG blending favors very high values of $\boldsymbol{\omega} \cdot \mathbf{t}$ for a single camera. High values represent pixels which are positioned around the center of the camera frame. These pixels are considered to be more reliable than the ones located on the borders of the frame. In practice, the majority of $\boldsymbol{\omega}$ have one dominant camera, *i.e.* these pixels will be around the frame center of only one camera. Thus, the AG blending should neutralize the effects of all other cameras by assigning them a very low confidence factor and keeping only the dominant camera, similar to NN blending. In situations when an $\boldsymbol{\omega}$ has more than one high value of $\boldsymbol{\omega} \cdot \mathbf{t}$, the confidence factors adapt the blending weights to use more than one contributing camera in the weighted average, resembling the Gaussian blending.

The proposed AG blending implementation does not visually differ from the approach consisting of creating two views. Furthermore, the standard deviation of the confidence factor can be manually adapted to obtain the best possible image quality.

An example of the AG confidence factor curve is shown in Figure 3.11, using σ_r set to $\frac{1}{30}$. The regions drawn over the curve depict the restrictions imposed on the Gaussian blending to obtain the AG blending. NN blending is applied in regions where the confidence factor is higher than 0.9, or almost 0, while Gaussian blending is applied in the transition region. This division reduces the influence of low-confidence over high-confidence pixels. Thus, the image mosaic is sharp in areas close to a single cameras' center, while the camera overlapping regions located on the periphery are blended using a Gaussian weight distribution.

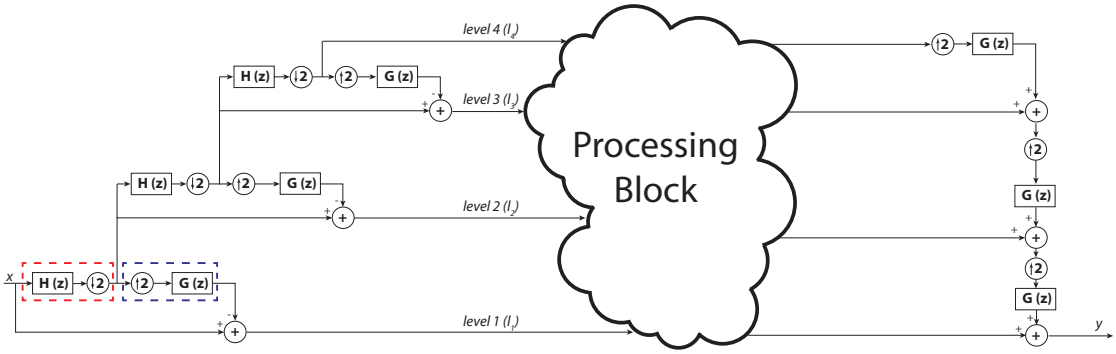


Figure 3.12: Laplacian pyramid decomposition and reconstruction with four levels. The top level (*level 4*) represents the coarse approximation, whereas the bottom level represents the details. The analysis (**H(z)**) and synthesis (**G(z)**) filters with internal downsampling are marked with red and blue dashed rectangles. They are marked only in the first level for clarity reasons. The processing block denotes operations on the decomposed pyramid. The LP reconstruction is performed on the processed pixels, on the right side of the figure.

3.6 Multi-Band Blending

Multi-Band Blending (MBB) [5] is based on a multiresolution decomposition of the original images and their blending across octave frequency bands. The images are represented using a Laplacian Pyramid (LP) [13], as it has perfect and simple reconstruction [60]. Several steps are performed to obtain the desired LP.

Laplacian pyramid is a multispectral and multiscale representation of an image, where each pyramid level contains one frequency band of the image. Image decomposition, processing, and reconstruction using a four-level LP is illustrated in Figure 3.12. Let x be the source image. The image is filtered using the analysis low-pass filter **H(z)** and downsampled by two, in both horizontal and vertical directions. The decimated image is then upsampled and filtered with the synthesis filter **G(z)**. The first level (l_1 in Figure 3.12) of the LP is obtained by subtraction of the interpolated image from the source x, and it represents the high-frequency content of the image, *i.e.* the details. The decimated image is also used as the source for the second level of decomposition. An L-level LP is created using $L-1$ repetitions of the mentioned principle.

The regions of overlap between images may be of an arbitrary shape. Thus, a mask should be created, defining the pixels which should be taken from the original image and their respective weight [13]. A binary mask is assigned to each image, where value 1 represents a pixel that should be taken from the selected image. This mask is further decomposed into a Gaussian Pyramid (GP), which is created by repetitive blur and downsample operations, *i.e.* each level of the pyramid is a low-pass version of the previous level.

Each frequency band of the LP is combined with the respective frequency band of the other LPs, *i.e.* other images. A weighted average is applied within the overlapped areas, which are proportional in size to the wavelengths represented in the band. Hence, when coarse features

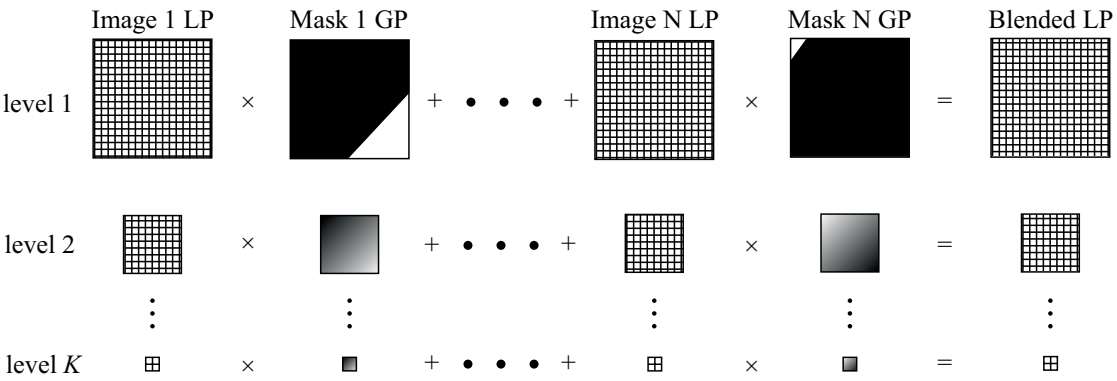


Figure 3.13: An illustration of multi-band blending of N images using decomposition into a K-level Laplacian Pyramid.

occur in the overlapping region, they are gradually blended over a relatively large distance without blurring or degrading finer image details in the neighborhood [13]. The weights are taken from the corresponding mask GP. In case of blending two images, image A and image B, the blending of one pyramid level is expressed as:

$$I(x,y) = I_A(x,y)\,w(x,y) + I_B(x,y)(1 - w(x,y)) \tag{3.15}$$

where I_A and I_B are pixel intensities and w is the pixel weight. Figure 3.13 illustrates MBB procedure of N source images into a single panorama.

3.6.1 Choice of Filters

Quality of the blended panorama relates to the chosen set of filters for MBB. In this thesis, only finite impulse response (FIR) filters are considered, since they are zeros-only filters and they can be efficiently implemented in hardware using convolution. A filter dataset of five low-pass filters is designed, and their performance is analyzed. The dataset consists of:

- 5-tap binomial filter [60]
- 5/3 LeGall filter
- 9/7 Daubechies filter
- Custom 9-tap maximally flat filter
- Custom 9-tap equiripple filter

The constraints on the custom-designed filters are driven by the hardware complexity and the frequency specifications of the first three filters. Hardware complexity and area utilization increase with the filter order. Additionally, high-order filters require large buffers for temporary

pixel storage. Thus, we derive constraints which fit specifications of all compared filters. The designed filters are maximum eighth order, the cut-off frequency is $\pi/2$, and passband and stopband attenuations are 0.03 dB and 70 dB, respectively. The designed filter is used to create an orthogonal filter set from which decomposition and reconstruction filters are used as **H(z)** and **G(z)**. These filters guarantee a perfect reconstruction of a single image, *i.e.* without blending. Thus, the loss of image quality in the reconstructed panorama pertains only to the stitching process and it is not a consequence of a faulty reconstruction.

Brown [5] and Burt [13] suggest to use the same filter for generation of the GP weight mask as for the LP. The use of the same filter simplifies the system and provides seamless blending results when the overall brightness level of the images does not differ significantly. However, when the images are taken with different exposure levels, the seam becomes noticeable [61].

To solve this issue, we propose the use of linearly distributed weights, instead of a Gaussian distribution. The proposed filter to create the GP of weights is:

$$H(z) = \frac{1}{5}(1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}) \tag{3.16}$$

The selected filter set for MBB was tested on a database of images, partly provided by A. Goshtasby from Image Fusion Systems Research [62]. Multiple shots of the same scene have been captured with different exposure times. More image sets were created by slightly shifting the image horizontally and diagonally, to emulate errors in registration that occur in real-life conditions. The benchmark database consisted of 30 different blended images and all of the input images had different exposure levels. One third had no registration errors, one third had horizontal shifts and one third had diagonal shifts. Objective image quality metrics were applied to compare the quality of blending. A set of objective metrics based on both perceptual visual quality and statistical properties of the image is determined. The set consisted of the No-Reference Blur Metric (NRBM) [63], the Edge Quality (QE) [64], and the naturalness index (NIQE) [65]. For color images, the values were measured on the luminance component. The photomosaics in these cases were created using the maximum number of pyramidal levels of decomposition. Table 3.1 shows the obtained results for four different scenes. The best result for a scene is marked in bold.

The results show that the 5/3 filter is superior to the compared filters in terms of edge sharpness (QE). Another edge quality metric, the NRBM, shows that the edges are less or evenly blurred compared to other filters. It can be also noted that the image *San Francisco* presents better results in all three comparisons, which is due to a high frequency nature of the image. For this type of images, the advantage of 5/3 biorthogonal filter pair can be noticed the best.

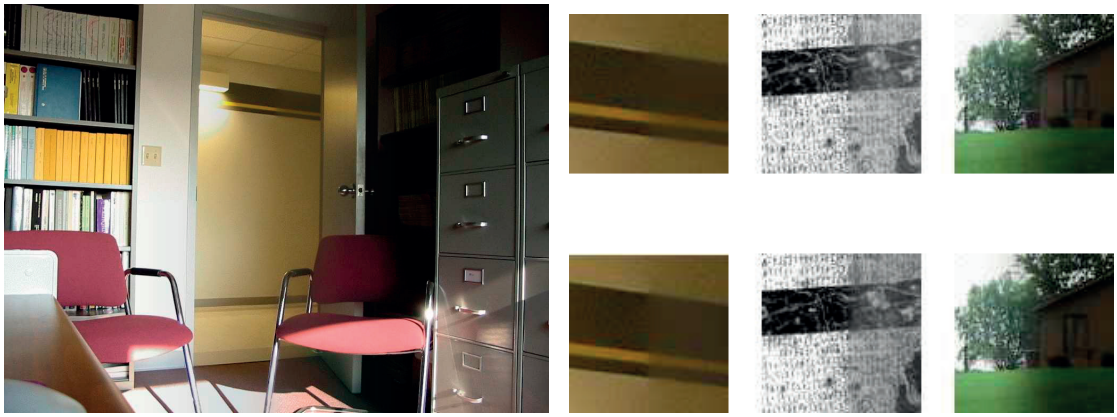
Figure 3.14a shows results of 5/3 blending of the *Room* image with two differently exposed halves. The left side of the image is taken from the brighter image, whereas the right side is from the darker. In Figure 3.14b, 5/3 filter effects are compared to maxflat in the neighborhood of the seam. When the source images have slight brightness difference, 5/3 filter almost

Table 3.1: Objective quality metric comparison

Image	Filter	Metric		
		NRBM	**QE**	**NIQE**
Room	**Binomial**	5.23	0.6568	**8.2810**
	5 / 3	**4.37**	**0.7368**	8.2840
	9 / 7	**4.37**	0.7119	8.2944
	Maxflat	4.59	0.7288	8.2842
	Equiripple	4.52	0.7302	8.3066
House	**Binomial**	4.11	0.6910	3.1445
	5 / 3	**3.85**	**0.7422**	3.1390
	9 / 7	3.86	0.7400	3.1435
	Maxflat	3.88	0.7403	**3.1365**
	Equiripple	3.87	0.7399	3.1370
Mountains	**Binomial**	3.62	0.8024	2.7472
	5 / 3	**3.61**	0.8575	**2.7273**
	9 / 7	**3.61**	0.8560	2.7441
	Maxflat	3.62	**0.8584**	2.7298
	Equiripple	3.62	0.8539	2.7278
San Francisco	**Binomial**	3.12	0.8170	2.2245
	5 / 3	**3.11**	**0.9471**	**2.2217**
	9 / 7	3.12	0.9376	2.2290
	Maxflat	3.12	0.9394	2.2488
	Equiripple	3.12	0.9369	2.2439

completely removes the visible seam (Figure 3.14b-left), which is not the case with the second-best filter (from Table 3.1 comparison), *maxflat*. The difference is more emphasized in regions where both high and low frequencies are present in the overlapping region. In the cropped part of the *San Francisco* [13] image (Figure 3.14b-middle), a difference in detail preservation can also be observed in the left, brighter part of the crop. Furthermore, the brightness difference of the background in the left and the right part of the image is slightly reduced.

Figure 3.15 shows luminance values of 50 pixels in one image row around the seam in Figure 3.14a. Pixel position 0 is the seam. The blue line represents luminance obtained by 5/3 and the red line represents *maxflat*. Luminance obtained by max is higher left of the seam and lower on the right, compared to 5/3. This is the consequence of brightness levels in initial images. Blending using 5/3 decomposition results in equalized luminance levels around the seam, which are more pleasant for the human visual system.



(a) Blended image using 5/3 filter

(b) Top row: blending using 5/3; Bottom row: blending using maxflat

Figure 3.14: Comparison of different filters used in multi-band image blending.

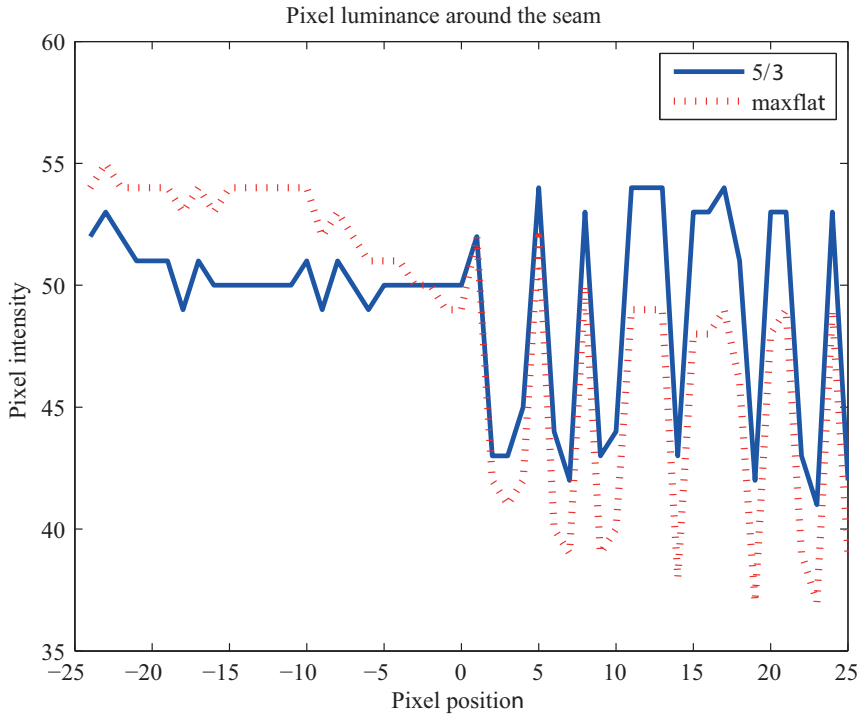


Figure 3.15: Pixel luminance around the seam using 5/3 and *maxflat* filters. The seam is located at position 0. The 5/3 filters result in smaller intensity difference between the left and the right side of the seam.

3.7 Conclusion

In this chapter, we showed how the image is formed on an image sensor, and we explained how the given geometrical relations can be used to stitch a panorama in a calibrated multi-camera system. We also presented a way to have a programmable FOV, while keeping the same panorama resolution. The differences in image brightness and ghosting are stated as the most noticeable artifacts in panoramas. We presented several real-time image blending methods to overcome this issue, and provided their image examples. The comparison shows that the Gaussian blending results in the improved image quality over the simpler real-time stitching and blending methods. Hence, the hardware implementation of this method will be given in Chapter 4, where the full design of a miniaturized multi-camera system will be presented.

4 Omnidirectional Multi-Camera System Design: Panoptic

This chapter explains the *Panoptic* camera, a real-time omnidirectional multi-camera system. The system is composed of a custom printed circuit board (PCB), with the full FPGA-based processing system on it. We explain the full processing pipeline, starting with the motivation and the system constraints for building such a system. It is followed by the system architecture, and block-by-block implementation details. We finish with discussion of the experimental results.

4.1 Introduction

In Chapter 2 we gave an overview of the currently available omnidirectional imaging systems. Majority of these systems provide very good image quality at the expense of speed. Whereas this trade-off is acceptable for photographers, applications such as autonomous navigation, telepresence, remote monitoring, or object tracking are strongly dependant on real-time processing. State-of-the-Art systems for these application are currently relying on a single camera solutions, due to lack of fast processing hardware on the market. The multi-camera systems provide omnidirectional capability, and they would certainly improve performance of the navigation or monitoring systems.

Multiple image stitching and blending is the most process intensive part of generating the image mosaic. They are usually implemented as a post-processing operation on a PC. Real-time operation is a very challenging problem. Hence, a GPU implementation or a dedicated hardware solution are often considered. Various existing GPU implementations of image mosaicing algorithms [10, 32] exist, but the problem with GPU implementations is in the scalability (see Figure A.1) and the limited bandwidth for data transfer from the camera system to the PC. On the other hand, FPGAs are widespread used platforms, that enable fast development and prototyping. The important advantages of FPGAs over GPUs are easier performance optimization and scalable designs. Processing speed of an FPGA system linearly drops with the increase of the image resolution, making the system scaling predictable. Considering these results, an FPGA-based system is chosen as the processing platform for the *Panoptic* camera.

Three main parts can be identified in *Panoptic* camera:

1. Image acquisition - Consisting of the image sensor and interfacing hardware

2. Image processing unit - An FPGA-based system implementing the omnidirectional vision reconstruction

3. User interface - Camera control and panoramic image display

4.2 Image Acquisition Module

The main component of the Image acquisition module is the commercial-off-the-shelf (COTS) image sensor. Since the portability of the system is desired, one criterion for the imager selection is its size. The second criterion is the camera's embedded image processing system-on-chip (SoC), which implements the fundamental raw image processing pipeline. Most of COTS modules designed for mobile phone integration meet these criteria, and we chose a low-cost PIXELPLUS PO4010N image sensor. The summary of the electrical characteristics of this sensor is given in Table 4.1. The full specifications are given in Appendix B. Figure 4.1a and Figure 4.1b show the image sensor with the connector, and the hemispherical dome where the sensors should be mounted.

Table 4.1: Main PO4010N characteristics.

Parameter	**Value**
Total Pixel Array	386 × 320
Effective Pixel Array	368 × 304
Pixel Size	3.6 μm× 3.6 μm
Filter	RGB Bayer color filter
Data Interface	8-bit parallel + synchronization + clock
Frame Rate	25 fps
Maximum Output Resolution	352 × 288 (CIF)

Figure 4.1c illustrates the internal block diagram of PO4010N sensor. The Image Signal Processing block is dedicated to raw image processing, *e.g.* Bayer-to-RGB conversion, white balancing, auto exposure, etc. It is controlled via Inter-Integrated Circuit (I2C) bus. The sensor outputs 8-bit parallel data, horizontal and vertical synchronization control signals, and the pixel clock used for synchronous transfer.

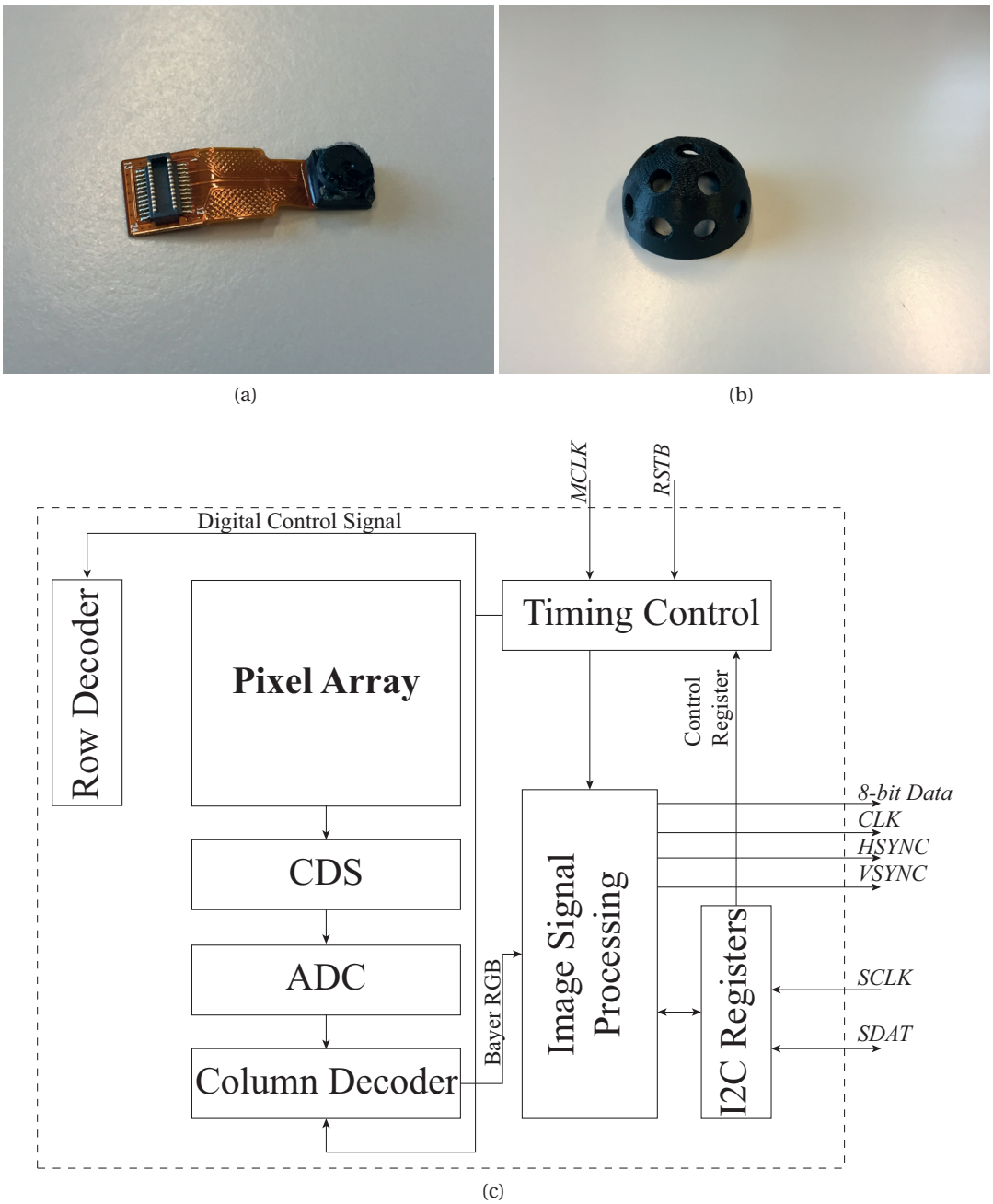


Figure 4.1: (a) PIXELPLUS PO4010N image sensor module, (b) the hemispherical dome for sensor placement, and (c) block diagram of PO4010N internal architecture.

4.3 System-level Analysis

The state-of-the-art multi-camera systems are built using COTS modules placed in an array, grid, or circular formation. The direct implementation of a multi-camera system consists of connecting all the imaging devices to a central processing unit in a star topology formation.

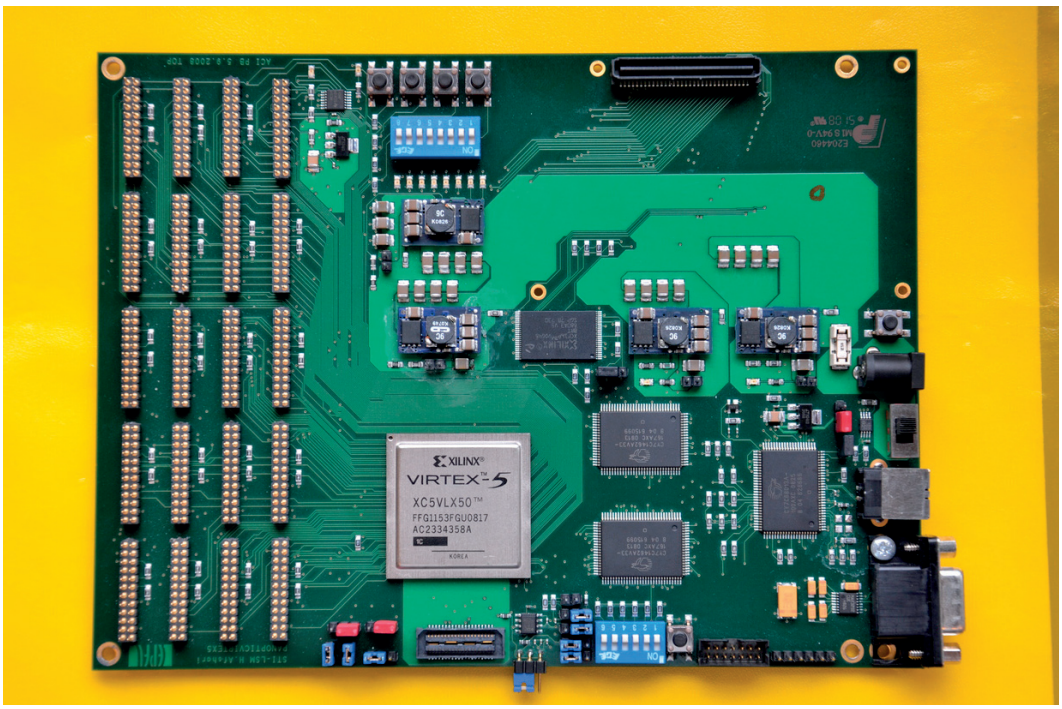


Figure 4.2: Virtex-5 processing board for *Panoptic* camera.

The responsibilities of the central unit are controlling the cameras, receiving the video streams, storing them into the memory, and implementing the processing algorithm for real-time panorama construction. This is called a *centralized* architecture.

A custom FPGA board (Figure 4.2) has been designed using a XILINX Virtex-5 XC5VLX50-1FF1153C FPGA as a core processing unit in order to capture and process the video streams produced by the cameras in real-time. The summary of the available resources on this is FPGA chip is given in Table 4.2.

Table 4.2: Summary of the available resource on Virtex-5 XC5VLX50.

Resources	**Number**
Logic Slices	7200
Block RAM capacity	1728 Kb
DSP blocks	48
User I/Os	560

This board directly interfaces with at most twenty PIXELPLUS PO4010N cameras, forming the aforementioned centralized architecture. The number of cameras connected to a single board is limited by the user I/O pin availability of the chosen FPGA chip. To support higher number of camera interfaces, multiple identical boards of the same kind can be stacked (see

Appendix D). The camera modules provide output data in 16-bit RGB format with selectable frame rate. The 16-bit pixels are in RGB565 format, *i.e.* red channel is coded with five bits, green with six, and blue with five. The cameras of the Panoptic system are calibrated for their true geometrical position in the world space, their intrinsic parameters, and the lens distortion coefficients [56]. Even though the camera calibration is precise within certain error bounds, the spherical arrangement of the cameras, *i.e.* diverging camera directions, emphasize parallax problems. Hence, appropriate blending algorithms are still needed for a seamless and ghost-free panorama.

4.3.1 System Memory and Bandwidth Constraints

The incoming video streams from the cameras have to be stored in the memory, and then later processed. The memory is segmented into $N_{cam} = 20$ segments of $C_w \times C_h \times BPP$ bits, where C_w and C_h are frame width and height, and BPP is the number of bits used for representation of one pixel. The needed memory capacity is:

$$M_{cap} \geq N_{cam} \times C_w \times C_h \times BPP \tag{4.1}$$

Another constraint related to the memory is its bandwidth. The bandwidth of the memory should sustain the aggregate number of access times for writing all the image sensor video streams and that of the application process within the defined fps timing limit. The required memory bandwidth to support real-time storage of frames for N_{cam} cameras is:

$$M_{BW} \geq FR \times N_{cam} \times C_w \times C_h \times BPP \tag{4.2}$$

where FR is the video frame rate in fps.

The required bandwidth for two different real-time frame rates and six different standard

Camera Resolution		FR [fps]	
C_w	C_h	25	30
320	240	30.5	36.6
352	288	40.5	48.6
640	480	122.5	147
1024	768	314.5	377.4
1920	1080	829.4	995.3
5120	3840	7864.3	9437.2

Table 4.3: Required memory bandwidth per camera in Mb/s.

Camera Resolution		**Memory Space** [Mb]
C_w	C_h	
320	240	1.23
352	288	1.62
640	480	4.92
1024	768	12.59
1920	1080	33.17
5120	3840	314.57

Table 4.4: Require memory space per camera in Mb.

camera resolutions are shown in Table 4.3. The bandwidth is calculated per camera and for $BPP = 16$ bits. As observed in Table 4.3, an increase in the camera resolution results in large bandwidth requirement. For example, a video stream from the state-of-the-art 20 Mpixels sensor occupies almost full bandwidth of the latest DDR3 memories.

Since twenty PO4010N cameras are envisioned for this board, the system must be able to store their video streams. Multiplying values from Table 4.3 and Table 4.4 by 20, the required memory bandwidth and capacity are:

$$\begin{aligned} M_{BW} &\geq 810 \ Mb/s \\ M_{cap} &\geq 32.4 \ Mb \end{aligned} \tag{4.3}$$

for the frame rate of 25 fps. Hence, two Zero Bus Turn around (ZBT) Static Random Access Memories (SRAM) with 36 Mb capacity and an operating bandwidth of 2.67 Gb/s are used on this FPGA board. One memory chip is used for storing the incoming image frames from twenty cameras, while the previous frame is fetched by the image processing core from the other memory chip. The two chips swap their role with the arrival of each new frame.

4.4 Top-level Architecture

The architecture of the FPGA is depicted in Figure 4.3. The FPGA design consists of five major blocks. The arrow lines depicted in Figure 4.3 show the flow of image data inside the FPGA. Image data streaming from the cameras enters the FPGA via the Camera interface block. A time-multiplexing mechanism is implemented to store the incoming frame data from all the camera modules into one of the single-port SRAMs. Hence, the Camera multiplexer block time-multiplexes the data received by the Camera interface block and transfers it to the Memory controller for storage in one of the ZBT SRAMs. The SRAMs are partitioned into twenty equal segments, one for each camera. The Memory controller interfaces with two external SRAMs available on the board at the same time. The Memory controller block provides access for storing/retrieving the incoming/previous frame in/from the SRAMs. As mentioned in the previous section, the SRAMs swap their roles (*i.e.* one is used for writing and one for reading) with the arrival of each new image frame from the cameras.

The Image processing and application unit is in charge of the image processing and basic camera functionalities, such as single video channel streaming, or all-channel image capture. This block accesses the SRAMs via the Memory controller, processes the fetched pixels according to the chose application mode, and transfers the processed image to the Data link controller. The Data link controller provides transmission capability over the external interfaces available on the board, such as the USB 2.0 link. Finally, the Camera controller block is in charge of programming and synchronizing the cameras connected to the FPGA board via I^2C bus.

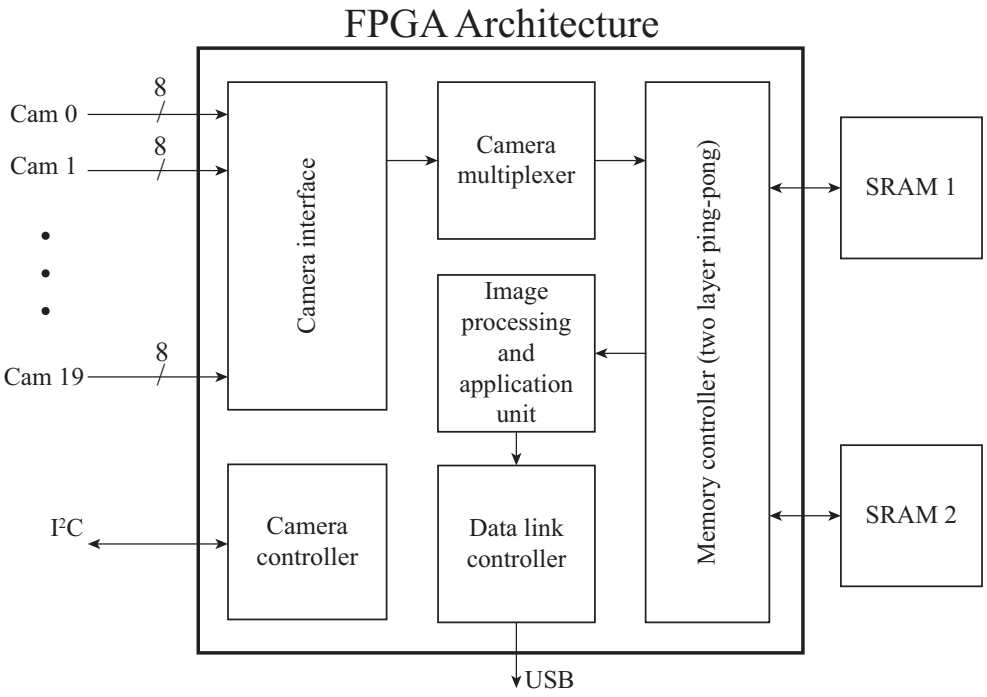


Figure 4.3: Top-level architecture of the *Panoptic* camera FPGA design. Arrows denote the data flow.

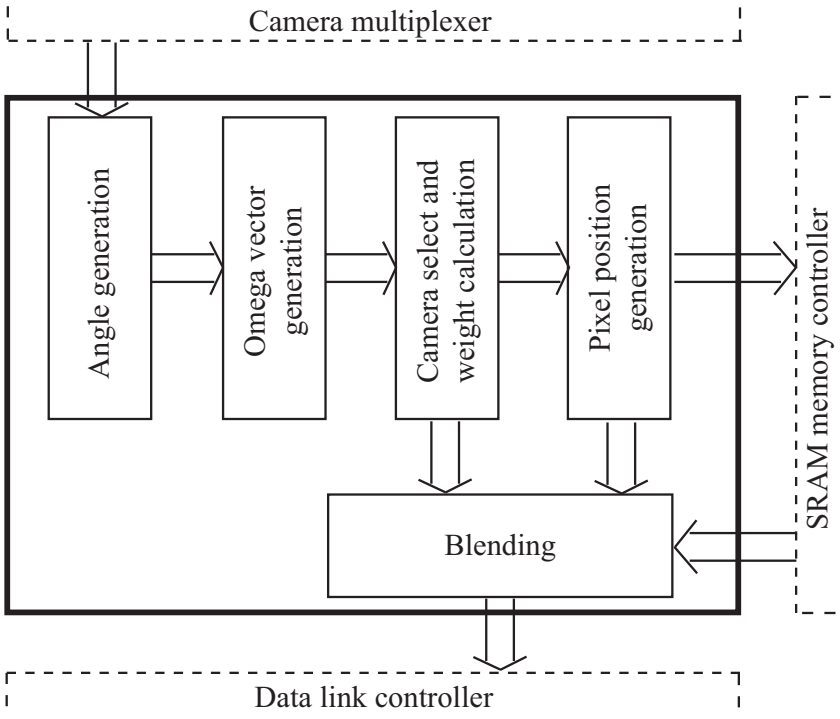


Figure 4.4: Block diagram of the Image processing and application block dedicated to the panorama construction. Each of the blocks within the bold rectangle is explained in detail in this chapter.

4.5 Implementation of the Image Processing Unit

The panorama construction algorithm is implemented inside the Image processing and application unit. The block diagram is shown in Figure 4.4. This image processing entity comprises five modules, which are thoroughly discussed in the following sections.

4.5.1 Angle and Omega Vector Generation

The Angle generation module generates the spherical coordinates $(\theta_\omega, \phi_\omega)$ of the $\boldsymbol{\omega}$ directions which are of interest for the reconstruction. It has the ability of generating angles for both equiangular and constant pixel density pixelization schemes from (3.6) and (3.8). The span and resolution of the output view is selectable within this module. It is possible to reconstruct a smaller portion of the light field with an increased resolution, due to the initial oversampling of the light field, *i.e.* the cameras record more pixels than displayed in the reconstructed image, as explained in Section 3.2.2. Hence, a more detailed image with a limited FOV can be reconstructed while keeping the same frame rate. Furthermore, higher resolutions can be achieved by trading-off the frame rate. The 13-bit representation of the angles leaves enough margin for a truthful reconstruction, considering the used CIF imagers and the total amount of the acquired pixels. The generic N-bit representation is as follows:

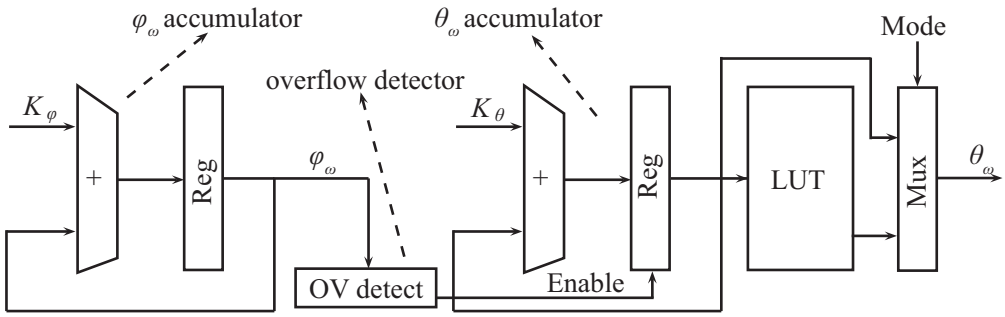
$$b_0 b_1 \cdots b_{N-1} = 2\pi \sum_{i=0}^{N-1} b_i \cdot 2^{-(i+1)} \tag{4.4}$$

Since the coordinate angles are represented by 13 bits, the maximum reconstruction resolution for a hemisphere is 16 Mpixels.

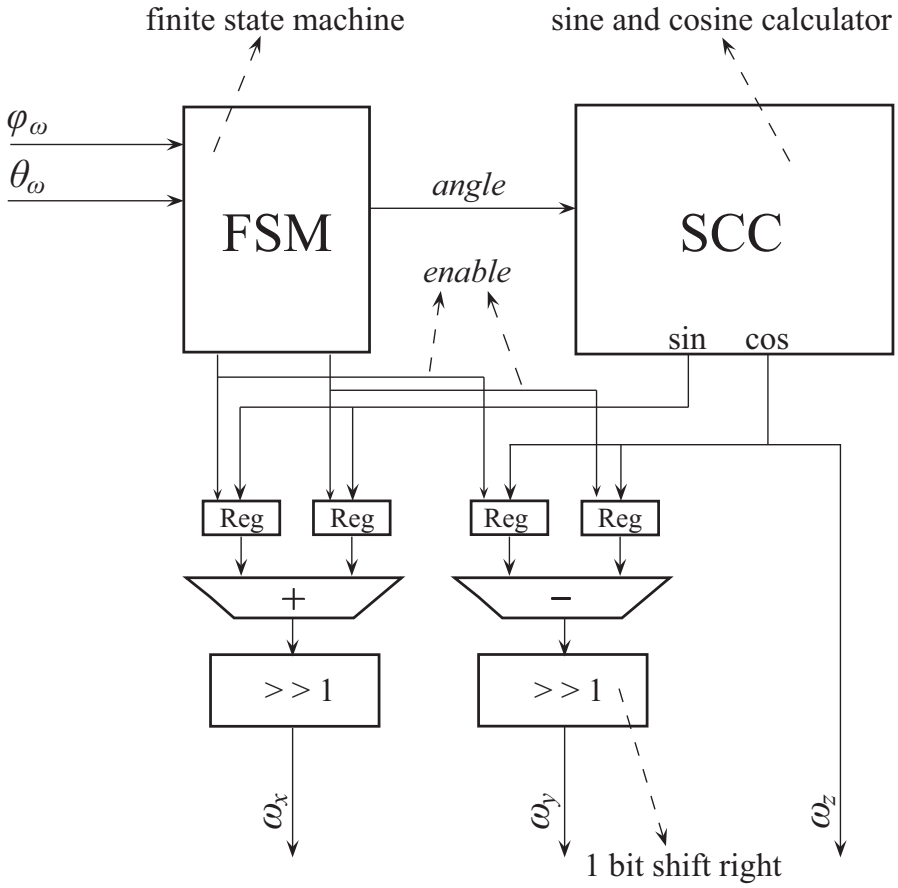
The angles θ_ω and ϕ_ω are implementable using an accumulator for each angle. To generate all possible combinations of θ_ω and ϕ_ω, the θ_ω accumulator increments when the ϕ_ω accumulator completes its full range cycle. This concept is shown in Figure 4.5a. The two accumulators can have different incrementing steps K_ϕ and K_θ. These incrementing steps define the resolution of the constructed omnidirectional vision. In addition, the limits ϕ_{min}, ϕ_{max}, θ_{min}, and θ_{max} are set for both angles. The FOV of the constructed panorama is determined through the configuration of these four parameters.

The Omega vector generation module calculates the radial unit vector pertaining to the spherical position $(\theta_\omega, \phi_\omega)$ received from the Angle generation module. The vectors are generated according to the following equation:

$$\boldsymbol{\omega} = \sin\theta_\omega \cos\phi_\omega \, \boldsymbol{x} + \sin\theta_\omega \sin\phi_\omega \, \boldsymbol{y} + \cos\theta_\omega \, \boldsymbol{z} \tag{4.5}$$



(a)



(b)

Figure 4.5: (a) ϕ_ω and θ_ω angle generation hardware. The LUT at the output stores the θ_ω values for the equal density discretization; (b) Hardware implementation of the $\boldsymbol{\omega}$ generation block.

Trigonometric functions $\sin(2\pi x)$ and $\cos(2\pi x)$ are used for the calculation of the $\boldsymbol{\omega}$ vector from the ϕ_ω and θ_ω angles. The implementation of the trigonometric functions sine and cosine has been the focus of direct digital frequency synthesizers (DDFS) for the past decades. Hence, many algorithms have been developed for the purpose of the implementation of basic trigonometric functions. Look up table (LUT) based algorithms [66], the CORDIC algorithm [67] and polynomial approximation based algorithms [68] are three widely accepted implementations. The LUT methods are the fastest and numerically the most precise, but they need memory storage. Since there is enough available BlockRAMs in the FPGA (Table 4.2), the LUT method is implemented in *Panoptic* camera.

The multiplication operations in (4.5) are replaced with their addition-based identities to reduce the number of needed multipliers, since their availability is limited in this FPGA:

$$\boldsymbol{\omega} = \frac{1}{2}(\sin(\theta_\omega + \phi_\omega) + \sin(\theta_\omega - \phi_\omega))\mathbf{x} + \frac{1}{2}(\cos(\theta_\omega - \phi_\omega) - \cos(\theta_\omega + \phi_\omega))\mathbf{y} + \cos(\theta_\omega)\boldsymbol{z} \quad (4.6)$$

The **x**, **y** and **z** components of the $\boldsymbol{\omega}$ are calculated utilizing a finite state machine (FSM), which provides three angles from (4.6) consecutively, and a single SCC module. The SCC module calculates and outputs the sine and cosine values simultaneously. Hence, the $\boldsymbol{\omega}$ is obtained by presenting the following angle combinations to the SCC module: $(\theta_\omega + \phi_\omega)$, $(\theta_\omega - \phi_\omega)$, and θ_ω, and combining their respective sine and cosine outputs in the correct order. An architectural view of the $\boldsymbol{\omega}$ vector generation module is shown in Figure 4.5b.

4.5.2 Camera Selection and Weight Calculation

The Camera select and weight calculation module determines which cameras contribute to the construction of the pixel in $\boldsymbol{\omega}$ direction. The camera c_i is considered to be contributing if the $\boldsymbol{\omega_j}$ is within it FOV α_i, *i.e.*

$$\omega_t = \boldsymbol{\omega_j} \cdot \mathbf{t_i} > \cos(\frac{\alpha_i}{2}) \quad (4.7)$$

Furthermore, this module computes the distance between the focal point projection and the virtual observer projection on the ω-plane, for each contributing camera c_i in direction $\boldsymbol{\omega_j}$, as expressed in (4.8):

$$r_{i,j} = |(\mathbf{q} - \mathbf{t_i}) - ((\mathbf{q} - \mathbf{t_i}) \cdot \boldsymbol{\omega_j})\,\boldsymbol{\omega_j}| \quad (4.8)$$

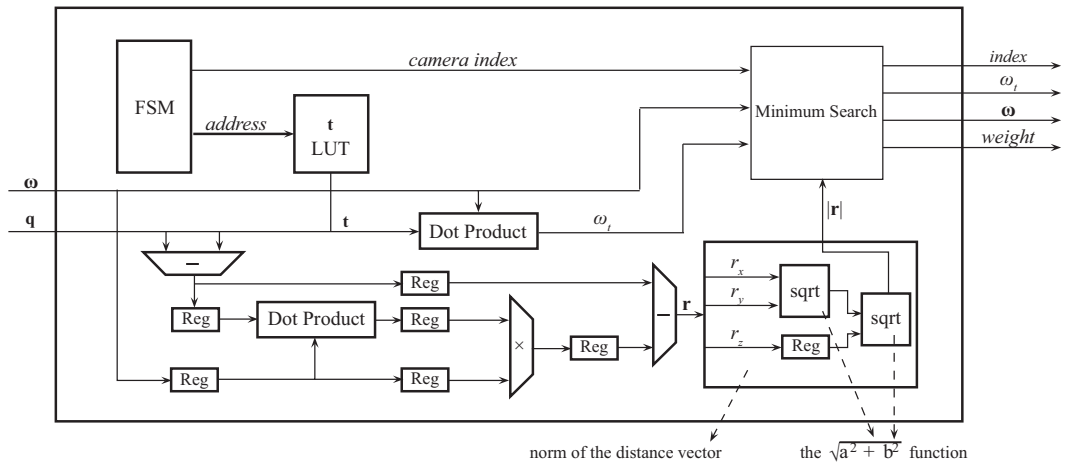


Figure 4.6: Block diagram of the Camera select and weight calculation module.

The pseudo-code describing the operation of this module is given in Algorithm 1, and its hardware implementation in Figure 4.6. For each pixel direction $\boldsymbol{\omega}$ that is received from the Omega vector generation module, this module sequentially provides the stored **t** values of all interfaced cameras, and calculates the ω_t dot product. The condition from (4.7) is checked in the Minimum Search sub-block. This architecture ensures consecutive checks of all *Panoptic* interfaced cameras, and their contribution to the selected $\boldsymbol{\omega}$ direction.

Concurrently, the distance vectors of the projected focal points of the cameras with the projected virtual observer point are calculated (4.8). The magnitude of the **r** is passed as the distance value to the Minimum Search sub-block. If the NN blending is chosen, Minimum Search sub-block finds the minimum distance r, and selects the corresponding camera as the only one at the output. If alpha or Gaussian blending is selected, the search is not necessary, *i.e.* indices and weights of all contributing cameras are provided at the output, sequentially. The weights are calculated using (3.12) and (3.13).

Dot Product and Square Root Sub-blocks

The dot product calculator of this module is implemented using a pipelined architecture consisting of three multipliers and two adders. Three multipliers are followed by a register stage. Two products are added together, whereas the third is again registered. Finally, the dot product is obtained after the final addition that is again registered. The pipelined architecture is used to boost the system performance and shorten the critical path that would be very long if both a multiplier and an adder were on it.

A norm of a 3D vector is also calculated using the pipelined architecture, by cascading two 2D vector norm calculators, *i.e.* $\sqrt{a^2 + b^2}$. The 2D norm calculator uses CORDIC algorithm, and it is implemented using only addition and subtraction operators [67].

Algorithm 1 Camera Select and Weight Calculation

$r_{\min} \leftarrow 1$
for all cameras **do**
  $\omega_t \leftarrow \boldsymbol{\omega} \cdot \mathbf{t}$
  $\mathbf{r} \leftarrow (\mathbf{q} - \mathbf{t}) - ((\mathbf{q} - \mathbf{t}) \cdot \boldsymbol{\omega})\,\boldsymbol{\omega}$
  if $(\omega_t > \cos(\frac{\alpha}{2}))$ **then**
    if blending == nearest_neighbor **then**
      if $(|\mathbf{r}| < r_{\min})$ **then**
        $r_{\min} \leftarrow |\mathbf{r}|$
        STORE camera_index
      end if
    else
      $r \leftarrow |\mathbf{r}|$
      index ← camera_index
    end if
  end if
end for
if blending == nearest_neighbour **then**
  $r \leftarrow r_{\min}$
  index ← camera_index
end if

4.5.3 Pixel Position Generation

The Pixel position generation module calculates the true pixel position in the image frame of the cameras selected in the Camera selection block. This goal is achieved using the pinhole camera model [54] to obtain the two-dimensional position (d_x, d_y) in the camera image plane, which is identified by the vectors **u** and **v**. The basic pinhole camera model equation is expressed in (3.5).

In reality, the mapping of a 3D scene onto an observed 2D plane of a camera image frame is a complex problem, which is only coarsely represented by (3.5). The intrinsic parameters of the camera are categorized in two classes, and characterize the mapping between a 3D scene and the observed 2D plane. The first class is the linear homography, defined by a 3×4 camera matrix, mapping of 3D points coordinates into 2D pixel coordinates [54]. The second class models the non-linear effects such as lens distortion. These parameters are estimated through the calibration process [56], stored for each camera in a LUT, and applied to results of the basic pinhole camera model as expressed in (4.11):

$$\begin{aligned} \omega_u &= \boldsymbol{\omega} \cdot \mathbf{u} \\ \omega_v &= \boldsymbol{\omega} \cdot \mathbf{v} \end{aligned} \tag{4.9}$$

$$\begin{aligned} d_x &= f \cdot \frac{\omega_v}{\omega_t} \\ d_y &= f \cdot \frac{\omega_u}{\omega_t} \end{aligned} \tag{4.10}$$

$$\begin{aligned} R^2 &= d_x^2 + d_y^2 \\ poly &= k_5 R^6 + k_2 R^4 + k_1 R^2 + 1 \\ d'_x &= poly \cdot d_x + 2k_3 d_x d_y + k_4 (R^2 + 2d_x^2) \\ d'_y &= poly \cdot d_y + 2k_4 d_x d_y + k_3 (R^2 + 2d_y^2) \end{aligned} \tag{4.11}$$

The obtained (d'_x, d'_y) values do not necessarily match with exact pixel positions in the camera frame. The nearest pixel position is chosen by simply rounding the (d'_x, d'_y) values, and getting the true (x, y) coordinates. Hence, the intensity value of the closest pixel (x, y) is chosen as the intensity value of (d'_x, d'_y). The same process is repeated for each camera frame, *i.e.* for all **u** and **v** vectors.

The Pixel position generation module interfaces with the SRAM memory controller to retrieve the pixel value of the contributing cameras upon calculation of the true pixel position. The camera index originating from the Camera select module is used to access the correct memory segment of the SRAM, *i.e.* the segment where the image frame of the selected camera is stored. The true (x, y) coordinates are mapped to their corresponding addresses in the memory segment used for storing the camera frame.

A detailed block diagram of this module is shown in Figure 4.7. The proposed architecture is pipelined, which allows streaming access to the true pixel positions. The pipeline registers are omitted for clarity purpose.

The Pixel position module also calculates the distance of the selected pixel in the image frame from the image center. This distance is represented as R' in Figure 4.7 and it is further used for Gaussian blending and the vignetting correction.

Sub-blocks of Pixel Position Generator

Hardware dedicated to the lens distortion compensation is marked with a box in Figure 4.7. Distortion compensation of the cameras is conducted on-the-fly in the proposed architecture rather than through the large transformation LUT for each camera, which demands additional memory space. A resource-efficient architecture has been achieved through factorization of similar terms, as shown in (4.10).

The arithmetic division operators are implemented using a four-stage iterative convergence method described in [69]. The polynomial function $f(R^2)$ that calculates the $poly$ term

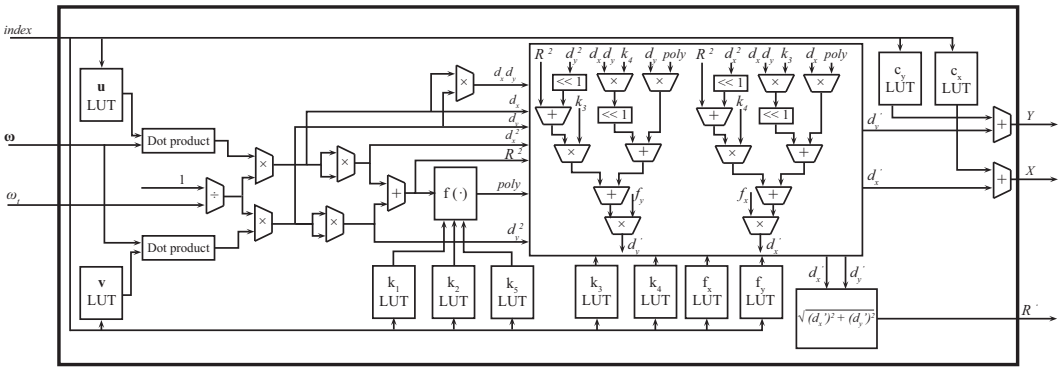


Figure 4.7: Block diagram of the Pixel position generation module. The pipeline registers are omitted for clarity.

in (4.11) is implemented using the Horner scheme [68] implementation of the polynomial functions. This method achieves higher precision in less number of iterations compared to the other alternative methods [68].

4.5.4 Image Blending

The Blending module conducts the final step of panorama construction. Three blending methods are currently available on *Panoptic* camera: nearest neighbor, alpha blending, and Gaussian blending.

The module receives the pixel values from all contributing cameras along with the blending weight (except when NN blending is selected), and the distance R', as shown in Sections 4.5.2 and 4.5.3. The weights are used to calculate the final pixel value using one of the equations (3.11), (3.12), or (3.13). As the final result, the block estimates a single light intensity value for each $\boldsymbol{\omega}$ direction.

A straightforward implementation of the blending algorithms is very resource-demanding. Multipliers and dividers would be required for each color channel separately. Thus, it is beneficial to share hardware resources which are common for all channels, *e.g.* implementation of the denominator in (3.13). For the implementation purpose, (3.13) is expressed differently:

$$\begin{aligned} \mathscr{L}(\mathbf{q},\boldsymbol{\omega}) &= \sum_{i \in I} a_i \cdot \mathscr{L}(c_i,\boldsymbol{\omega}) \\ a_i &= \frac{w_i \cdot G(R'_i)}{\sum\limits_{k \in I} w_k} \end{aligned} \tag{4.12}$$

where a_i is the normalized weight of the i^{th} contributing camera, and $G(R'_i)$ is the vignetting correction factor. The vignetting correction factors are pre-calculated and stored in a LUT.

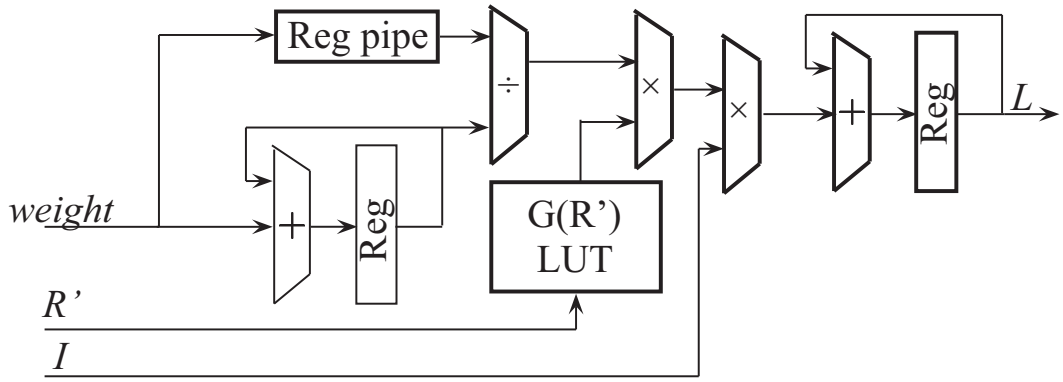


Figure 4.8: Block diagram of the Image Blending module.

As explained in Section 3.3, this factor is addressed in LUT using only the pixel distance from the camera's optical center. The pseudo-code of the blending block and its hardware implementation are given in Algorithm 2 and Figure 4.8. I_{RGB} represents color intensities of the contributing pixels in the algorithm notation.

Algorithm 2 Blending

if blending == nearest_neighbor **then**
  $\mathscr{L}_{RGB} \leftarrow I_{RGB}$
else
  $w_{acc} \leftarrow \sum_{k \in I} w_k$
  for all $i \in I$ **do**
    $a_i \leftarrow w_i \cdot \frac{1}{w_{acc}}$
    $a_i \leftarrow a_i \cdot G(R'_i)$
  end for
  for all color_channels **do**
    $\mathscr{L}_{RGB} \leftarrow \sum_{i \in I} I_{RGB} \cdot a_i$
  end for
end if

4.6 Experimental Results of the *Panoptic* System

A *Panoptic* hemisphere of diameter $2r = 3\ cm$ is built using a 3D printer. It can accommodate fifteen PO4010N cameras, arranged on three floors. The hemisphere populated with cameras is positioned on top of a plexiglas structure that is attached to the designed PCB, as shown in Figure 4.9. The camera modules are connected to the digital interfaces on the PCB using flexible PCBs, and they are operated at 25 fps.

The architecture presented in this chapter was developed in VHDL for the target FPGA. The developed firmware conducts all mathematical processing using 16-bit fixed-point precision. The sufficient bit precision of the mathematical operations was verified through the MATLAB

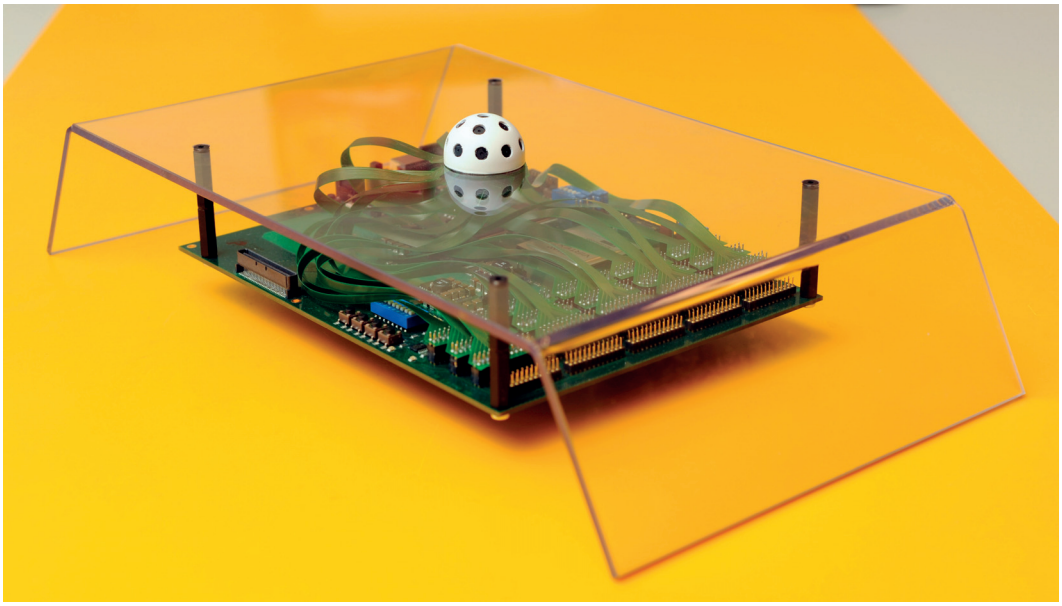


Figure 4.9: The fully assembled *Panoptic* camera system.

model. The system control unit shown in Figure 4.3 is implemented using a Xilinx Picoblaze soft-core 8-bit microcontroller.

The firmware was targeted and successfully tested for operation at 133 MHz f_{clk} frequency. The total latency of the system is 132 clock cycles, which is less than 1 μs, using 133 MHz frequency. The power consumption of the FPGA board, when its firmware is in full operation, was measured at 5.05 Watts. The aspect ratio of the reconstructed panorama is kept at 1:4 to be support 90° × 360° FOV without any additional distortions. Hence, the firmware is able of constructing an omnidirectional view with the resolution of 0.25 MPix (256 × 1024) at the rate of:

$$FR = \frac{f_{clk}}{N_{cam} \cdot C_w \cdot C_h} = 25.3\, fps \tag{4.13}$$

which is the frame rate of the cameras providing the input video streams.

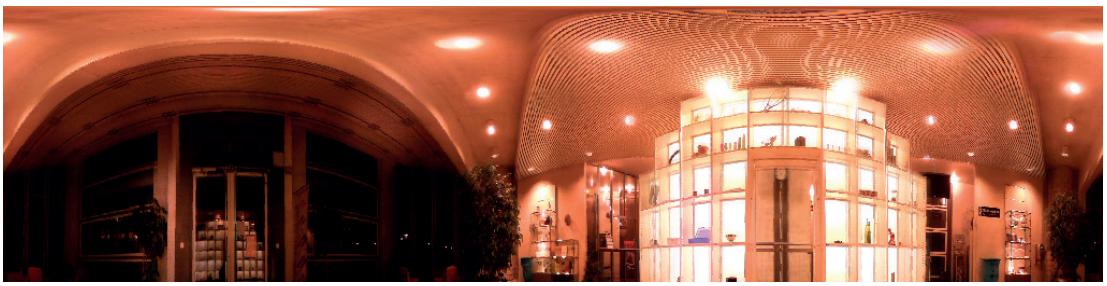
The blending methods presented Sections 3.4 and 3.5 were separately implemented on the FPGA in order to compare their resource utilization. The summary is presented in Table 4.5. Gaussian and Adaptive Gaussian blending infer additional LUTs and multipliers compared to NN and alpha blending. This is observed through the increase of the used BlockRAMs and logic slices. However, the increase of resource usage compared to the alpha blending and NN is very small and is not an influential factor in the overall utilization.

Four image captures from real-time panoramic reconstruction are shown in Figure 4.10. The Gaussian blending with $\sigma_d = 100$ was used.

(a)

(b)

(c)



(d)

Figure 4.10: Image captures from the *Panoptic* video stream. The Gaussian blending with $\sigma_d = 100$ is used in all images. The scenes are from different locations around EPFL campus.

Table 4.5: *Panoptic* Virtex-5 FPGA resource utilization comparison

Blending	Nearest Neighbor	Linear	Gaussian	Adaptive Gaussian	
Resource			Used		Available
Slices	4070	4653	4607	4816	7200
Slice Registers	9351	10069	10127	10196	28800
BlockRAMs	17	17	21	22	48
DSPs	37	47	48	48	48

4.7 User Interface and Display

The external host for the developed prototype is a PC that interfaces with the developed FPGA platform. The role of the PC is restricted to video display and parameter selection in the FPGA firmware. No additional image processing is conducted on the PC side. A Graphical User Interface (GUI) is developed in the PC for controlling the *Panoptic* camera. The GUI also offers video display, image capture and recording capability. More information concerning the GUI software can be found in Appendix E.

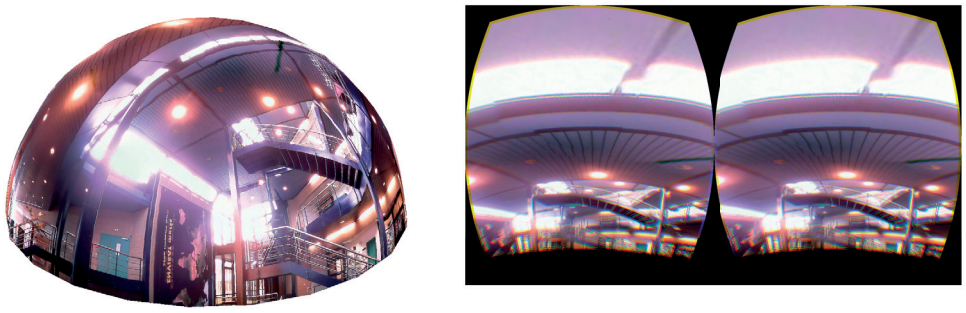
The *Panoptic* camera can be used as a perfect example of a telepresence system. Unlike the virtual reality systems, where users are transported to a virtual scene, telepresence allows users to be in another location in the real world, *e.g.* videoconferencing. Among the benefits of videoconferencing, we can say it lowers the travel requirements, improves dialog efficiency and allows the mobility-impaired people to visit distant places. Instead of using the narrow FOV cameras, we can achieve a better telepresence experience.

The setup used to test and demonstrate the system consists of the *Panoptic* camera, a PC, and the virtual reality headset Oculus Rift HMD. The camera generates omnidirectional images at a resolution of 1024 × 256 pixels and transmits them via USB 2.0 link to the PC.

The developed GUI supports Oculus Rift, and creates a virtual environment in order to display the hemispherical image. This virtual environment is created using the OpenGL API and consists of a user controlled camera and a large overhead hemisphere, onto which the image is projected. The omnidirectional image is used as a texture for the virtual hemisphere. The camera rotates according to the sensor data received from the head-mounted display.

Figure 4.11a shows the textured virtual hemisphere from the side. When using the application with the head mounted display, the user viewpoint is in the middle of the sphere. Figure 4.11b shows the application in normal use with the HMD.

In order to ensure a high frame rate at all times, the application receives new omnidirectional images in a secondary thread. Thanks to the multi-threaded implementation the rendering frame rate is independent from the USB speed, as well as the camera frame rate. This is important for the streaming functionality, in which the frame rate can vary.



(a) (b)

Figure 4.11: (a) The textured OpenGL hemisphere showing a captured image, viewed from the side. (b) The client application generating the left- and right-eye view for the head-mounted display.

4.8 Conclusion

The current trend in building multi-camera systems is to use multiple commercially available camera modules. *Panoptic* , a real-time multi-camera system is presented in this chapter. *Panoptic* is implemented using the centralized processing approach of the multi-camera system. The full design was detailed, including the camera choice and specifications, constraint analysis, a real-time hardware implementation of the omnidirectional view construction, and finally, the display options. The omnidirectional snapshots were shown, together with the hardware resource utilization of several image blending techniques.

It is shown that the processing demand is rather high even for modest panorama resolution and low resolution image sensors. The centralized approach for implementing real-time applications in multi-camera systems is not efficient in terms of processing. As an alternative, workload distribution and parallel implementations are encouraged for achieving high resolution and high frame rate reconstructions. Providing embedded workload distribution and parallelism capability in a multi-camera system requires innovation at the architectural level. In the next chapter a novel distributed approach is introduced for the realization of high-performance multi-camera systems using the very high resolution cameras.

5 Towards Real-Time Gigapixel Video

In the previous chapter, we explained the design flow and the full hardware implementation of *Panoptic* , a real-time omnidirectional multi-camera system. *Panoptic* is a miniaturized system consisting of fifteen cell phone cameras providing 256 × 1024 resolution output. In this chapter we will present *GigaEye II* , a modular high-resolution multi-camera system, capable of achieving gigapixel resolutions.

5.1 Introduction

Panoptic camera presented in Chapter 4 is a scalable system, as shown in Appendix D. However, the scalability is reflected in stacking multiple processing boards, and blending the full omnidirectional image at the end. There are three main disadvantages of such approach: (1) each of the stacked boards reconstructs the full panoramic frame, creating a significant data overhead, (2) the resolution of individual cameras is limited due to capacity of the used ZBT SRAMs, and (3) reaching very high resolutions is not possible in real time.

The problem of data overhead can be solved by using the distributed processing approach instead of the centralized one. Seyid et al. [31] designed an interconnected network of smart cameras, where each camera represents a node in a mesh network, and processes only the pixels in its own FOV. Each camera in this system has a dedicated frame storage SRAM that still limits the maximum resolution of the camera, due to small capacities of static memories. Furthermore, the data traffic patterns and the network latency limit the total system's throughput, effectively lowering the frame rate at high panorama resolution.

In *GigaEye II* , the aforementioned limitations are resolved as follows:

1. The distributed approach is used by dividing the system into M clusters of N cameras, where each cluster processes only its FOV.
2. High-capacity double data rate 3 (DDR3) dynamic RAM (DRAM) is used for frame storage within each cluster, allowing scalability on the camera level.

3. Resolution is easily increased by including additional clusters to the system, which are connected either to the central unit, or an intermediate board, via a high-speed link. Thanks to the distributed processing, the processing time is not affected by the addition of the new cluster.

In the rest of the chapter, the full *GigaEye II* system will be presented in a similar form to *Panoptic* .

5.2 Camera Module Design

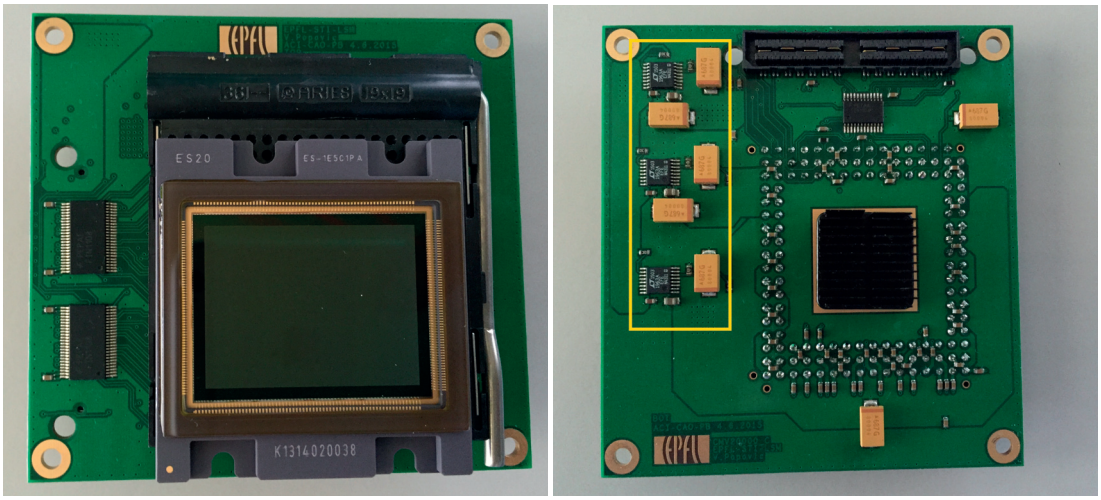
In order to achieve high acquisition resolutions with a reasonable number of image sensors, a CMOSIS CMV20000 color sensor is chosen. The sensor outputs 20 Megapixels at 30 fps frame rate. The summary of the sensor's specifications is given in Table 5.1. The full specifications are given in Appendix C.

Table 5.1: Main CMV20000 specifications.

Parameter	**Value**
Total Pixel Array	5124 × 3844
Active Pixel Array	5120 × 3840
Pixel Size	6.4 μm× 6.4 μm
Filter	RGB Bayer color filter
Output Format	12-bit
Data Interface	16 LVDS data channels + 1 LVDS control line + 1 LVDS DDR output clock
Frame Rate	up to 30 fps
Sensor Control Bus	SPI

The sensor headboard PCB is designed according to the CMOSIS guidelines, and it can be seen in Figure 5.1. The sensor is placed on a zero insertion force (ZIF) socket for easy placement and removal. In order to achieve the maximum frame rate, an f = 480 MHz clock has to be provided externally. In order to keep the signal integrity for such high frequencies, the high-speed SAMTEC differential cables are used. Furthermore, the power dissipation of the DC/DC converters (marked in Figure 5.1b), and the image sensors is high, which leads to overheating of the chips, the image sensor and the PCB. Hence, the used DC/DC converters must be large and with high thermal resistance, and a heat-sink should be placed on the PCB at the back of the sensor. The first designed version of this PCB did not consider the heat dissipation, and parts of it were overheating over the manufacturer's limits, as shown in Appendix F.

The analog-to-digital converters (ADC) in the sensor provide a 12-bit digitized value for each pixel that is serialized and sent to the output. The sensor's outputs consist of sixteen low-



(a) (b)

Figure 5.1: (a) The front side of the sensor headboard. CMV20000 color sensor is installed on a ZIF socket. Two visible chips are LVDS repeaters to drive the signal through a cable to the processing board; (b) The back side of the PCB showing power distribution part marked in a yellow rectangle, a SAMTEC connector for multi-gigabit transmission and a heat sink to cool down the PCB.

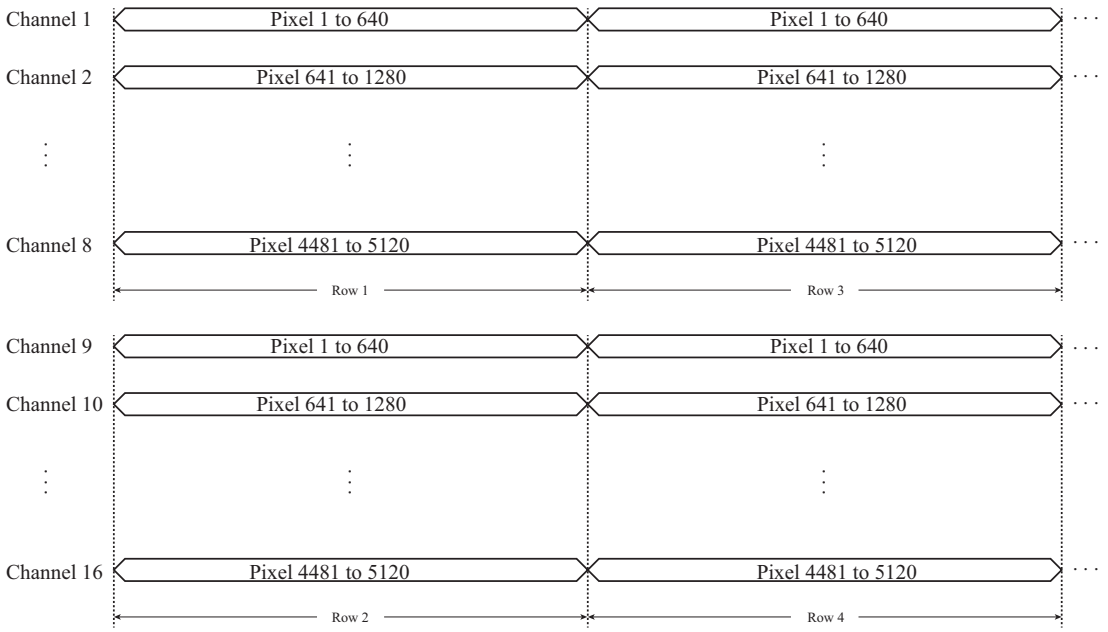
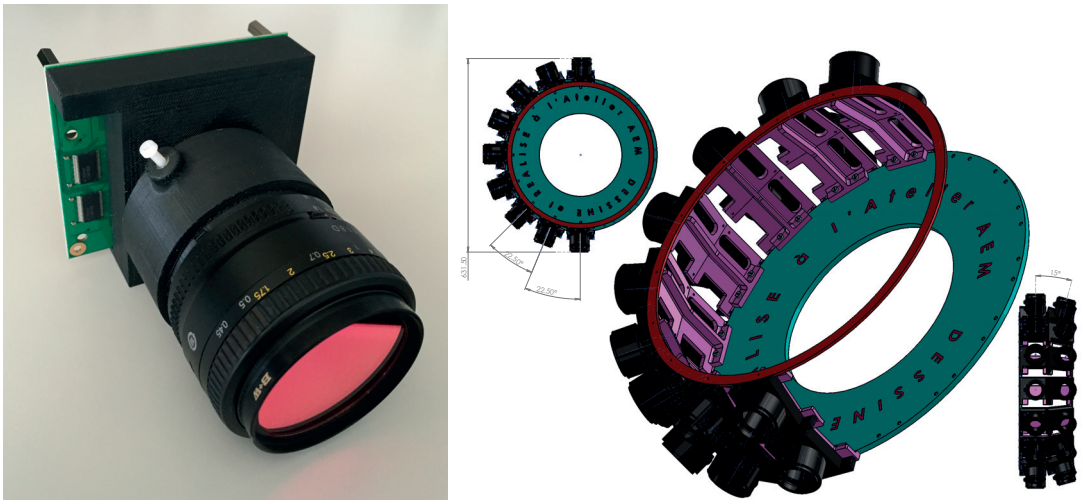


Figure 5.2: Pixel mapping for sixteen output channels of CMV20000.



(a) (b)

Figure 5.3: (a) Assembled camera module with a 50 mm Nikon lens and an external infrared filter, and (b) the technical drawing of the *GigaEye II* structure.

voltage differential signaling (LVDS) channels, which send sixteen different pixels. The frame is divided into eight vertical strips 640 pixels wide, and two horizontal blocks where one block consists of even rows, and the second one of odd rows. The intersection of a vertical strip and a horizontal block forms one output channel, as shown in Figure 5.2.

The full camera module is shown in Figure 5.3a. A lens holder is fabricated using a 3D printer, and it includes an adjustable lens mount cylinder. The cylinder is used to adjust the flange distance, *i.e.* the distance between the sensor and the mount ring. The flange distance is made adjustable in order to have the flexibility in the choice of lens, image sensor socket, and to compensate for potential imprecision of the 3D printing process. For this camera module, a Nikon F-mount 50 mm lens is chosen. Since, the CMV20000 is the color sensor without any infrared (IR) filtering, an external IR filter is placed on the lens itself.

5.3 System Design

Opposite to the miniaturized *Panoptic* camera, the camera modules in *GigaEye II* do not cover the whole hemisphere. A partial 3D model of the *GigaEye II* structure is shown in Figure 5.3b. The structure currently allows placement of thirty-two cameras in two rings, with the top ring inclined by 15° upwards. The structure can be upgraded by adding more rings on top of the current one, to construct a full hemisphere if needed. For the purpose of this thesis, sixteen cameras are installed and tested in two different arrangements: 1) all sixteen cameras installed on the bottom ring, covering a full 360° view, and 2) sixteen cameras in two rows of eight, with increased vertical FOV.

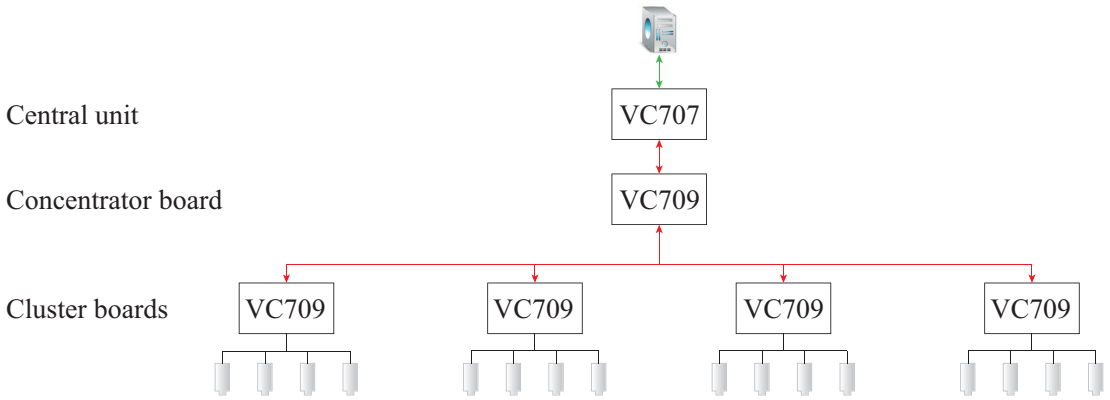


Figure 5.4: The full system diagram of *GigaEye II* . The system consists of three main layers: the cluster boards, the concentrator board, and the central unit. The cluster and concentrator boards are XILINX VC709 development kits, and the central unit is VC707. The red lines denote the high-speed optical links between the boards, and the green line corresponds to the user interface, such as HDMI, USB2, and UART.

Multiple FPGA boards are required to process the large amount of incoming data from the cameras. The diagram of the full *GigaEye II* system is shown in Figure 5.4. The system is divided into three layers:

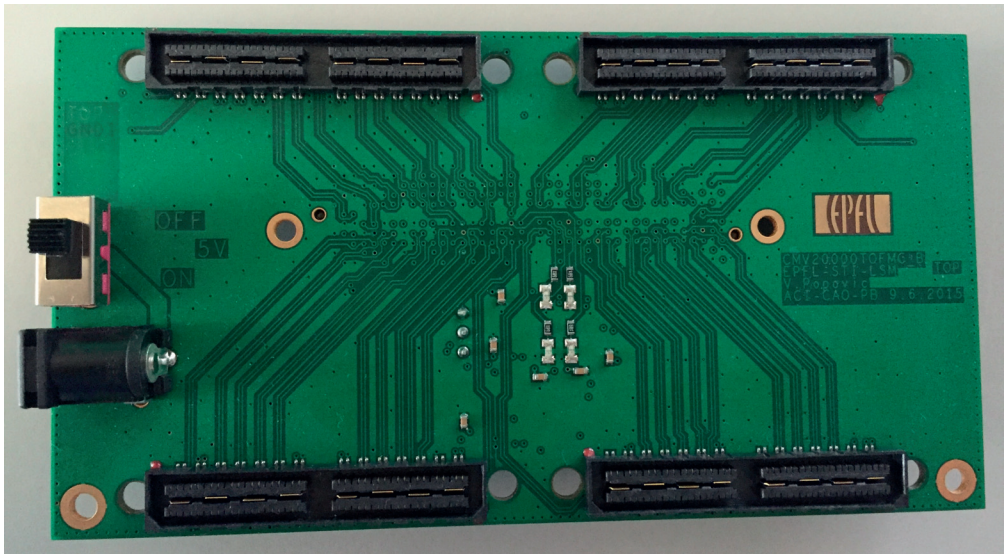
1. Cluster boards - Four cameras form a cluster connected to a single FPGA board. The designed processing system inside the FPGA creates a partial panorama and forwards only that part to the layer above.

2. Concentrator board - Four cluster boards are connected to the concentrator board that stores all partial panoramas, and merges them into a single composite frame.

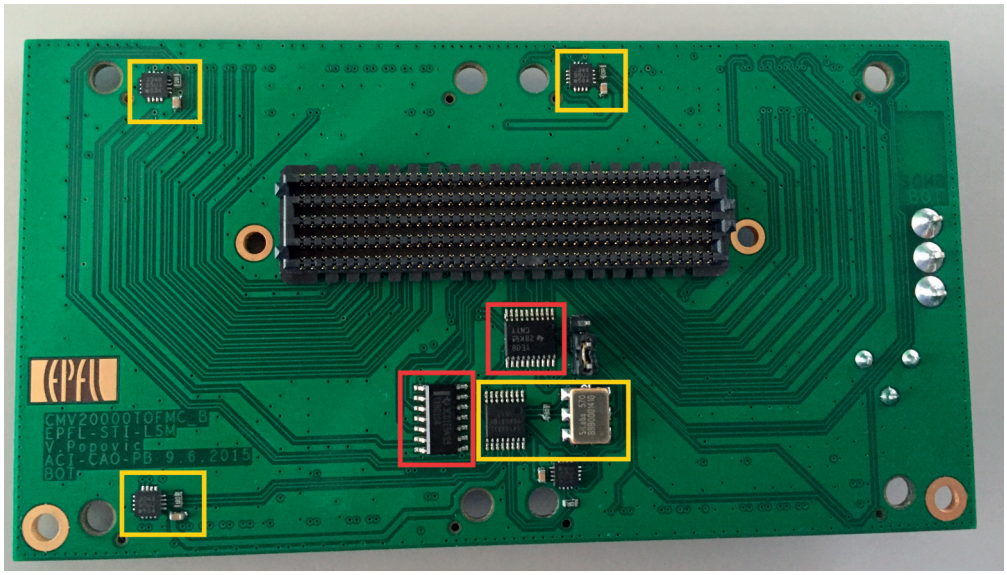
3. Central unit - The central FPGA board provides an interface towards the user (PC), external displays, and to the concentrator board.

The XILINX Virtex-7 FPGA is chosen to be the main processing core of each layer, since it is the latest generation FPGA providing lots of processing capabilities. The cluster layer and the concentrator layer are implemented on the VC709 development kits. The VC709 board, shown in Figure 5.6a consists of two DDR3 modules, which makes it suitable for the implementation of the image acquisition and image processing algorithms. Similar to *Panoptic* , during one frame time, one memory module is dedicated to storage of the current frame, whereas the second module is accessed by the panorama construction hardware.

Each cluster board is equipped with an FMC expansion connector. The design of the VC709 board provides 160 user available pins on this connector, which is enough to connect only four CMV20000 cameras. Thus, the design decision on the number of cameras in each cluster is driven by the number of available connections on each FPGA board. An interconnection PCB



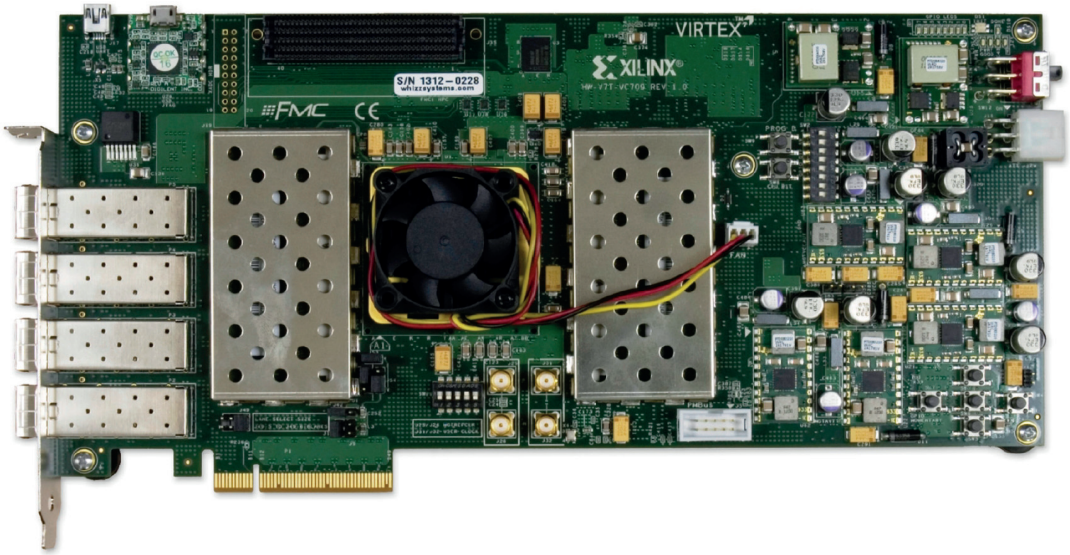
(a)



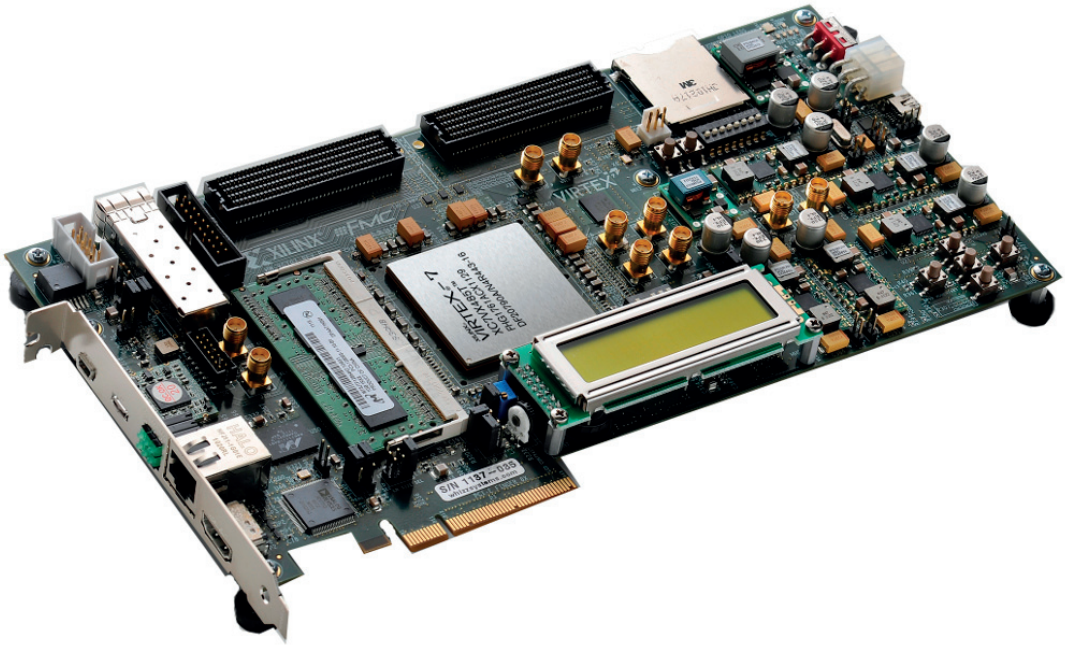
(b)

Figure 5.5: The FMC interconnection PCB. (a) The top view shows the high-speed SAMTEC connectors used by cameras, and (b) the bottom view shows the FMC connector in the center, and the clock distribution components marked in yellow rectangles.

is designed to connect all four cameras to a single FPGA board, and it is shown in Figure 5.5. The PCB includes four SAMTEC connectors for camera interface, an FMC connector for the FPGA expansion, and the clock distribution hardware. Since the cameras require a very high-speed $f = 480$ MHz clock that must be transferred through a cable, the strong, low-skew clock drivers are placed on the clock tree for each camera. Furthermore, because of the high data rates, all LVDS lines are length matched, and impedance matched to $Z = 100\Omega$.



(a)



(b)

Figure 5.6: Two Virtex-7 development kits used as (a) the cluster and the concentrator processing boards (VC709), and (b) the central unit (VC707).

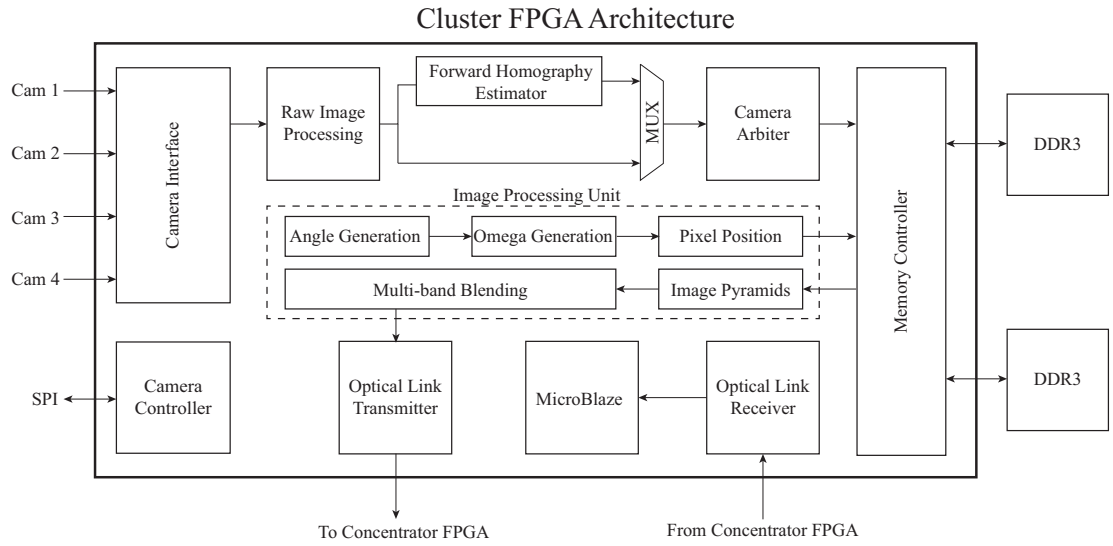


Figure 5.7: Top-level architecture of the *GigaEye II* cluster FPGA.

The connection between the cluster and concentrator boards is realized using the 10 Gb/s optical link that is available on the VC709 board. This link is shown in red color in Figure 5.4. However, the VC709 board is targeted at intensive processing, and not for user interfacing. Hence, it is not suitable for the implementation of the central unit.

The central unit is implemented on a VC707 development kit, shown in Figure 5.6b. Apart from an optical link interface, the VC707 includes an HDMI port, a USB2 interface, as well as a low-speed serial UART connection. The responsibilities of the central unit include control of the other system boards, indirect control of the cameras via the cluster boards, receiving the full reconstructed panoramic video streams, and providing it on the HDMI output (green line in Figure 5.4).

Opposite to the centralized architecture implemented on *Panoptic* , the distributed architecture shown in Figure 5.4 provides enough processing power to process this amount of pixels in real time.

The following three sections will give a detailed description of each processing layer, and their internal architecture.

5.4 Cluster Processing Board

5.4.1 Top-level Architecture

The top-level architecture of the cluster FPGA is depicted in Figure 5.7. Compared to *Panoptic* top-level architecture shown in Figure 4.3, the cluster board is more complex due to several differences between camera modules, such as the raw Bayer output of CMV20000, the high

frame resolution, and the different blending method in the Image Processing Unit.

The arrow lines depicted in Figure 5.7 show the flow of image data inside the FPGA. The serialized image pixels streaming from the cameras enter the FPGA via the Camera Interface block, which is in charge of synchronizing the FPGA with the cameras, deserializing the data, and multiplexing sixteen channels of each camera. The Raw Image Processing Block performs a *de facto* standard processing pipeline that includes noise reduction, white balancing, Bayer demosaicing, RGB blending, and contrast and brightness control.

The *GigaEye II* has two main operation modes. The first one is the full resolution mode, which is similar to the *Panoptic* camera. All pixels acquired by the cameras are stored in the memory when this mode is used. The reconstruction hardware in the Image Processing Unit generates the memory addresses of the needed pixels, which are then fetched from the memory. While this method is acceptable for a simple panoramic video construction, it is not possible to implement any additional functionality due to random access to DDR3, and the drop in memory performance in such conditions. The performance drop is explained in the following subsection.

The second operation mode is the high-performance mode. This mode allows additional operations to be implemented along the panorama construction, since it estimates and stores only the pixels really needed by the desired application. Hence, the Image Processing Unit requests the pixels in a stream mode, which is the highest performance mode of any DRAM. The Camera Arbiter implements a time-multiplexing mechanism to store all the incoming frame data from all the camera modules into one of the DRAMs. Similarly to *Panoptic* , the Memory controller interfaces with two external memories on the FPGA board at the same time. The Memory controller block provides access for storing/retrieving the incoming/previous frame in/from the DRAMs.

The minor part of the Image Processing Unit rests the same. The Angle Generation and the Omega Generation Blocks calculate the 3D coordinates of each $\boldsymbol{\omega}$. The Angle Generation block does not generate angles for the full panoramic view, but only for the selected region observed by the four connected cameras. Unlike Gaussian blending used in *Panoptic* , *GigaEye II* implements the MBB algorithm presented in Section 3.6. Thus, there is no need to calculate the weights based on the distance of the camera projection from the virtual observer. The Pixel Position block remains the same.

The MBB algorithm requires decomposition of all four camera frames into LP. The Image Pyramids block is dedicated to that purpose, as well as for generating the GP of weights for each pixel of each camera. Finally, the Multi-band Blending block merges the four image pyramids into a single one, and sends the data to the concentrator FPGA board via the optical link transmitter.

5.4.2 Camera Interface

The Camera Interface block deserializes the camera LVDS lines and converts them in a 12-bit parallel pixel data. In order to successfully deserialize the input data, each LVDS channels has to be trained independently. The goal of the training is bit and word alignment of all LVDS channels. The bit alignment is done to ensure that a channel is sampled in the center of its eye diagram. It is achieved by adding delay taps to the input data path of the channel, using the embedded differential input buffers of Virtex-7. The word alignment ensures that the first bit of all channels is sampled at the same clock edge. It is achieved by rotating the received parallel word until the output matches the training sequence.

The goal of the bit alignment is to place the sampling point in the ideal position for each channel. This is done by adding delay taps to the data input line, which shifts it relative to the sampling clock. One bit period consists of two regions: a stable and an unstable region. Sampling in the stable region guarantees that the correct data will always be sampled, regardless of the number of samples that are acquired. In the unstable region, the chance of sampling correct data is not 100%. The unstable region exists due to the non-ideal rise and fall times between two bits, and due to jitter on data and clock lines.

Determining if the sampling point is in a stable or unstable region is done by sampling N 12-bit sequences. If the sampled sequence has N times the same value, the sampling point is considered to be stable. The number of samples N should be high enough for a decent statistical coverage, and it is set to $N = 128$.

Finding the start and end of the bit period is done by continuously determining if a selected sampling point is stable or unstable, adding delay and checking again. At the start of the bit alignment routine, the relative position of clock and data is not known. Therefore, the routine will shift the sampling position until it finds an unstable point. From this point on, the unstable region starts. The training controller will continue shifting the sampling point in the same direction until it finds a stable sampling point. The only remaining point to be found is the end of the stable region. Hence, the controller shifts the sampling point until the next unstable sampling point is found. When this point is found, the controller knows the boundaries of the stable bit period and it can place the sampling point in its center.

The serial data channel contains a continuous stream of bits, and it is impossible for the receiver to know the position of the first bit of a 12-bit word. The goal of the word alignment is finding the position of the first bit in the word. Word alignment is done by continuously sampling the 12-bit training word that the sensor transmits in the training mode. If the sampled 12-bit word does not match the expected training word, the sampling point of the first bit of a word is moved by 1 bit period. This is done until the training word is matched.

Figure 5.8 illustrates the architecture of the Camera Interface. Apart from the four deserializers, the interface includes channel multiplexers for each camera. CMV20000 outputs sixteen pixels at a time. The pixels are from two consecutive row, with 640 columns offset between them.

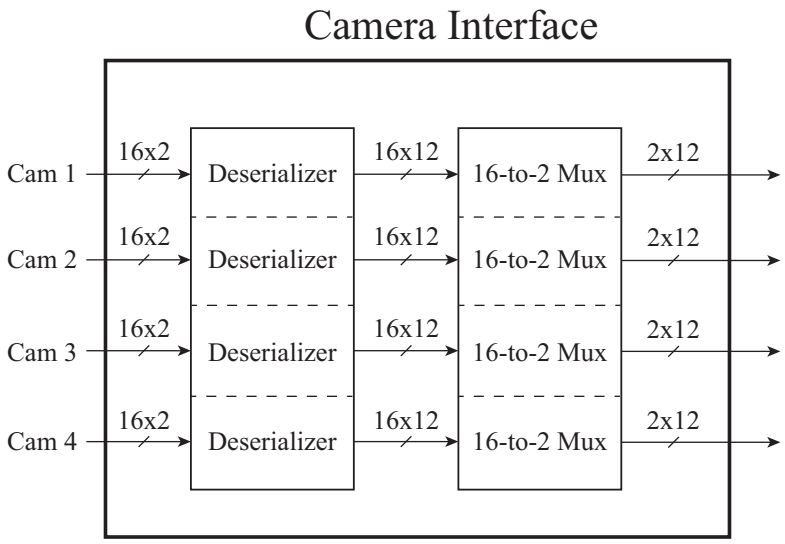


Figure 5.8: Block diagram of the Camera Interface block showing deserializers and camera channel time-multiplexers.

The goal of 16-to-2 multiplexer shown in Figure 5.8 is to efficiently reorder the incoming pixels, and create memory addresses for each one of them.

The operation of this multiplexer is driven by parameters of the system. The memory data bus is 512-bit wide. The 12-bit raw pixels are converted into RGB in RGB101010 format, *i.e.* ten bits are used for each of the color channels. Thus, each pixel can be stored as a 32-bit value, with only two bits of overhead data. These system parameters allow sixteen pixels to be written to DDR3 at the same time. In theory, it is possible to store the pixels as they come, *i.e.* to store sixteen arriving pixels. However, this creates a memory addressing problem since the mapping of pixel position to the memory address is non-linear, and requires a resource-demanding hardware.

Hence, a special multiplexer is implemented that buffers eight arriving pixels in each channel. Once eight pixels are buffered, the multiplexer reads sixteen pixels, eight from each row from the same vertical strip. Eight vertical strips are served in a round-robin manner until all the input buffers are read. A memory address jump of 640 pixels is included between each vertical strip. The procedure is repeatedly performed for all pixels in the frame. With this pixel arrangement, the memory addressing is linear, since the neighboring pixels are stored in the adjacent memory addresses. Hence, the hardware that translates the pixel coordinates from Image Processing Unit to the memory address is simple, straightforward, and has low-resource utilization.

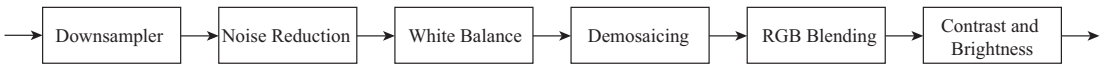


Figure 5.9: Block diagram of the implemented functions in the Image Processing Pipeline.

5.4.3 Raw Image Processing Pipeline

In general, the Raw Image Processing pipeline is responsible for taking the raw data from the image sensor and generating an image that can be displayed on a screen. Different processing blocks can be included in this pipeline [70] depending on the used camera and its on-chip processing options. The pipeline implemented in *GigaEye II* for CMOSIS CMV20000 sensor is shown in Figure 5.9.

The optional downsampler block downsamples the image that comes from the sensor, by removing pixels within a row or a column, in order to achieve the desired output resolution.

The noise reduction block is responsible for reducing the noise produced by the image sensor. The implemented block reduces the noise from three different sources: ADC offset, fixed-pattern noise (FPN), and photo response non uniformity. Hence, this sub-block is composed of three stages.

In the first stage, a fixed offset, defined by a 12-bit input is subtracted from every pixel. This correction essentially sets the dark level of the sensor.

In the following stage, the corresponding 12-bit FPN correction value is subtracted. The FPN is a light-independent noise and corresponds to the standard deviation of an averaged image. It is estimated by taking multiple images in the dark, with short exposure, and averaging them.

In the final stage, a gain correction is applied to each pixel in order to correct PRNU. The PRNU is caused by the difference in light sensitivity of the each pixel, *i.e.* pixels have a different light response curve. It can be obtained by taking several light gray (about 50 % of the sensor swing) images, averaging them and subtracting the FPN and the offset.

The final output value of the noise reduction block can be expressed as:

$$I_{corr} = (I_{raw} - I_{offset} - I_{FPN}) \cdot g_{PRNU} \tag{5.1}$$

where I_{corr} is the final corrected pixel intensity, I_{raw} is the raw data from the sensor, I_{offset} is the ADC offset, I_{FPN} is the FPN correction value, and g_{PRNU} is the PRNU gain correction multiplier.

Thanks to correlated double sampling (CDS) done in the pixel, the pixel-to-pixel FPN/PRNU is quite small. More noticeable is FPN/PRNU caused by the column amplifiers on the sensor. Hence, it wass chosen to have a per column FPN/PRNU correction, which also reduces

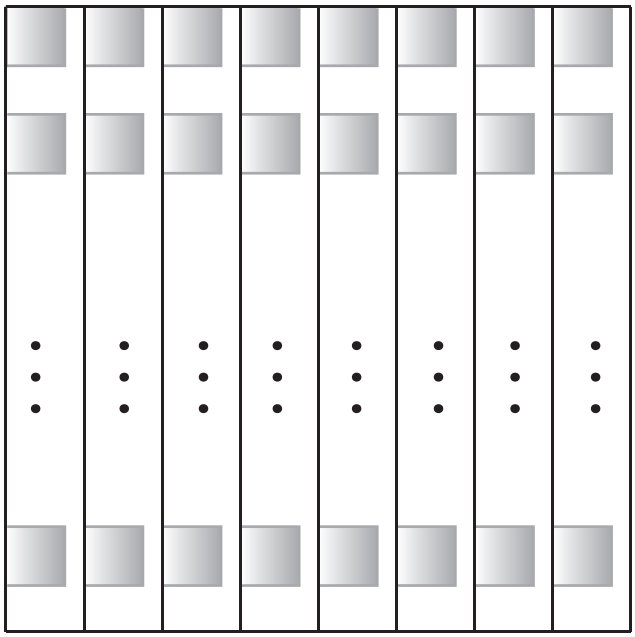
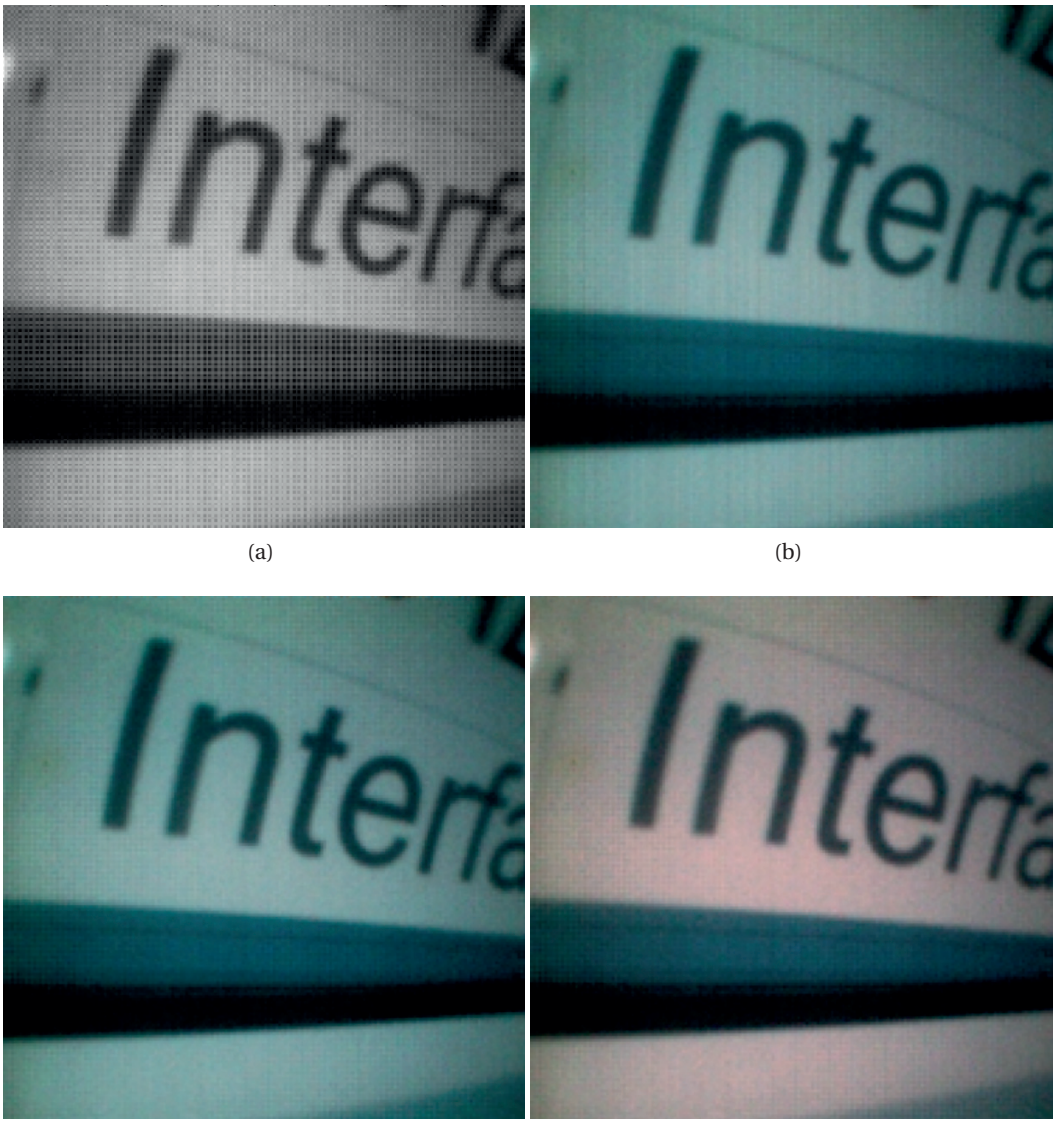


Figure 5.10: The distribution of sub-frames considered during white balancing. The shaded regions of size 512 × 512 pixels are chosen for an efficient white balancing hardware implementation.

memory requirements for storing the correction coefficients. The FPN and gain values are obtained by addressing a BlockRAM in the FPGA, with the column index of the corresponding pixel.

The White Balancing block adjusts the red, green, and blue values so that the white color appears white in the final image, in any lighting condition. This is done by multiplying the red, green and blue values by different gains factors. The White Balance block is composed of two main parts.

The first part is a circuit that determines the gain values that should be applied to the next frame, based on the values of the current frame. The gains are calculated by computing the means of red, green and blue, and then dividing them by the smallest mean among the three in order to have gains larger or equal than 1. Calculating the means can be a very intensive process, especially since division is required. The gain estimation is simplified by calculating the mean values among only a subset of pixels. The number of sub-frames, rows and columns in which the mean is calculated is chosen to be a power of 2, since the dividers can be replaced by a shift right operation. The sub-frames are chosen in such a way that the pixels considered for the mean calculation are well distributed across the image. An illustration of sub-frame distribution is shown in Figure 5.10. After obtaining the mean value for each color channel, the gains are computed by assuming that the green gain is default 1, and dividing the green mean value by the red and blue means. This is called the “Gray world” method, and in this implementation it provides the color adjustment with respect to the green color.



(a) (b)

(c) (d)

Figure 5.11: An example of the effect of Raw Image Processing on the final image. (a) The raw input image shown in grayscale, (b) demosaiced image, (c) denoised and demosaiced image, and (d) denosied, white balanced and demosaiced image.

The second part of the White Balancing block multiplies the gains with the red, green, and blue pixels. This results in the white balanced image, such as the one shown in Figure 5.11d.

The next block in the processing pipeline is the demosaicing. Since the camera sensor is covered by the Bayer color filter, each pixel receives either red, green, or blue component of light spectrum. Thus, it is necessary to demosaic the image, *i.e.* to interpolate the missing colors. There are several algorithms for Bayer demosaicing, and the chosen one is as follows. For each pixel, the color components that are filtered out are interpolated from the neighboring

pixels having the desired component. The component that is let through the filter remains the same, without considering any of the neighboring pixels. The results of the demosacing are shown in Figures 5.11b - 5.11d.

The RGB Blending block compensates for the fact that different image sensors produce different RGB values for the same color. Tuning this pipeline stage involves creating a blending matrix to convert the sensor RGB color space to a standard RGB color space. This is done by multiplying each RGB pixel by the matrix obtained by taking images of a calibration ColorChecker chart.

The Contrast and Brightness control is the block that performs a multiplication and an addition/subtraction. First, each color component of a pixel is multiplied by the same contrast coefficient, and then the brightness coefficient is added to each component.

Since the optimal contrast and brightness vary based on the particular lighting conditions, as well as upon user preference, these parameters are implemented so that they are dynamically adjustable by the user.

5.4.4 Forward Homography Estimator

Section 5.4.1 introduced the second operation mode of *GigaEye II* called high-performance mode. This mode allows implementation of more than one applications, by reducing the memory load and storing only the needed pixels for the desired applications. This is realized using the Forward Homography Estimator (FHE).

Real-time homography is usually perceived as an inverse problem thanks to a rather simple reconstruction pipeline. It is shown in Section 4.5 that the inverse homography is suitable for the application such as panorama construction, since the input images are be stored in memory before performing the actual reconstruction. For each desired pixel in the panorama, the most appropriate pixel can be found in the original images. The mapping function is either determined by using runtime calculations [55] or pre-calculated and stored in LUTs [60].

However, timing constraints become very tight when the desired output resolution is high. The image processing systems can hardly meet the real-time constraints of 25-30 fps when reconstructing high-resolution images. Hence, we introduce the forward homography as a possible solution to this problem, which has already been used in stereo image rectification systems [71, 72, 73] where the same real-time constraints apply.

The forward homography solves the issue of the system constraints, such as memory bandwidth, since the correct destination is calculated for each input pixel. Hence, only the necessary pixels for the final reconstruction are stored in memory, thus reducing the required bandwidth. The state-of-the-art forward homography systems pre-calculate the destination coordinates offline, and store them in registers of the processing system.

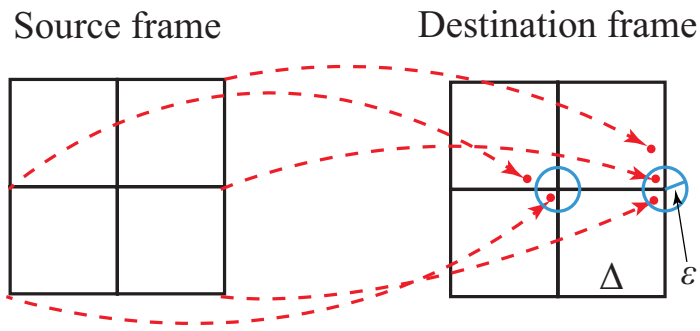


Figure 5.12: Possible results of the proposed forward homography estimator. Intersections of black lines represent source and destination grid pixels, whereas red lines and dots are projections and projected pixel locations. The blue circles of radius ϵ mark the area in which the projected pixels are considered correct. When more than one pixel is within the blue circle, value of the last pixel in the incoming pixel stream is assigned to the pixel in the circle's center.

However, LUT size linearly increases with respect to the input image resolution. CMOSIS CMV20000 sensor outputs 20 Mpixels frames, and LUTs become too large for the FPGA's internal memory. Compressed LUT methods [73] may partially solve this problem, but the peak-signal-to-noise ratio (PSNR) drops significantly in the presence of large differences between input and output image resolution. These differences are also observed in the majority of modern cameras, whose high-resolution images are usually displayed on the 2 Mpixels displays.

Estimating the forward homography in real-time is not a trivial problem. The system should determine the final pixel position in a panoramic image, based only on the pixel coordinates in the original frame. The problem arises due to non-integer values of the mapped pixel coordinates, as illustrated with red dotted lines in Figure 5.12. When observing homography as an inverse problem, it is easy to scan through the desired pixel grid and choose the closest pixel from the original frame. Forward homography processes a pixel stream, and the system can determine the closest position on the destination pixel grid. However, it cannot determine if the current pixel in the stream is the closest to the destination pixel, since it cannot predict the positions of pixels that have not been processed yet. Thus, pixels that are mapped to the same position are overwritten and the last pixel that appears in the stream will be considered as the correct one. Hence, the PSNR can be significantly decreased. This problem is even more emphasized in high-performance mode of *GigaEye II* , where hundreds of pixels from the original 20 Mpixels frame are mapped into a single one in the Full HD panorama.

We developed a new homography estimation algorithm to overcome this issue. By recalling the image formation illustration and equations from Section 3.1, equation (3.2) expresses the projection of a point in the 3D space onto the image plane. The first step of the algorithm is to back-project the pixels from the image frame. Each pixel $\mathbf{d}$ is back-projected into a line $\mathbf{l}$ in a 3D world that includes the focal point (projection center) $\mathbf{O_c}$. The line is illustrated in

Figure 3.1 and expressed by the inverse of (3.2):

$$\mathbf{l} = M^{+} \begin{bmatrix} \mathbf{d} & 1 \end{bmatrix}^{\top} \tag{5.2}$$

where M^{+} denotes a Moore-Penrose pseudoinverse of the projection matrix. Thus, back-projection results in a set of lines (light rays), where each one of them contains the focal point of the lens. In order to obtain 3D world coordinates, we should define a back-projection surface. We choose a unit sphere, $|r| = 1$, for the purpose of panorama construction. Back-projection onto the unit sphere is performed by normalizing the line $\mathbf{l}$ by its L^2 norm, and transforming Cartesian (x, y, z) coordinates into spherical (θ, ϕ, r), where θ is the polar angle, ϕ is the azimuth, and r is the radius:

$$\begin{aligned} \mathbf{X}_{sph} &= \mathbf{l} \,/\, ||\mathbf{l}||_2 \\ \theta &= \arccos\left(\mathbf{X}_{sph}(z)\right) \\ \phi &= \arctan\left(\mathbf{X}_{sph}(y) / \mathbf{X}_{sph}(x)\right) \\ |r| &= 1 \end{aligned} \tag{5.3}$$

If $\mathbf{X}_{sph}$ is the back-projected pixel, and $\mathbf{X}_s$ is the sampling point on the hemispherical pixel grid, we define a projection error as:

$$e = ||\mathbf{X}_{sph} - \mathbf{X}_s||_2 \tag{5.4}$$

Afterwards, we find a threshold value ϵ, such that at least one distinct back-projected pixel $\mathbf{X}_{sph}$ exists for each sampling point $\mathbf{X}_s$ with the error $e \leq \epsilon \leq \frac{\Delta}{\sqrt{2}}$, where Δ is the distance between two pixels on the hemisphere.

Five outcomes are possible in a 2D homography depending on the source and destination pixel positions. The simplest one is a 1-to-1 mapping when each pixel from the source frame maps to one in the destination frame. Furthermore, 1-to-0 and 0-to-1 are also possible, when the source pixel does not have a corresponding pixel in the destination frame, and vice versa. These three mappings are trivial cases and they will not be considered in the analysis.

Complex 1-to-N and N-to-1 mappings occur when destination and source frames are over-sampled, respectively. We resolve the 1-to-N mapping in the estimation algorithm by choosing the optimal ϵ. The optimal ϵ value ensures a distinct source pixel for each ϵ-neighborhood in the destination frame. Hence, a 1-to-N mapping is replaced by N 1-to-1 mappings.

Oppositely, ϵ value should be kept as low as possible in order to efficiently resolve the N-to-1

mappings. The problem that arises in forward homography is that the N^{th} pixel in the stream is considered as the correct, unless a full mapping is stored in the internal LUT or the external memory. Thus, the optimal ϵ is the lowest value that ensures 1-N resolving. Minimizing the ϵ value in the proposed estimation also reduces the number of pixel candidates to $M < N$. The benefit of this reduction is two-folded: 1) increased chance of choosing the optimal pixel, and 2) smaller error and higher PSNR when non-optimal pixel is chosen.

The homography between the image frame and unwrapped hemispherical surface is shown in Figure 5.12. Intersections of black lines represent pixel positions on the respective grids. Red dashed lines illustrate homography between two frames, and red dots are projected pixel positions in the destination frame. Distances between red points and the closest intersection of black lines is the corresponding error e. The blue circles mark the ϵ-neighborhood in which projected pixels are considered as potential candidates for the final pixel value.

Figure 5.12 illustrates two different cases of N-to-1 homography. Two source pixels on the left are mapped to the vicinity of a single destination pixel. The ϵ-neighborhood around the central destination pixel is set such that only one of the mapped pixels is inside the circle. Hence, the central destination pixel is given the value of the pixel inside the circle, which is indeed the closest projected pixel. In another situation, three pixels on the right side of the source frame are projected around one destination pixel. Two projections are inside the ϵ-neighborhood and one of them will be chosen as the destination pixel, *i.e.* the last one read out from the sensor.

Internal architecture of the FHE is shown in Figure 5.13. Subtraction of the camera center point position (x_0, y_0) translates the image frame origin to the frame center. Different row vectors of the matrix M^+ are provided to the dot product blocks, which evaluate the matrix multiplication in (5.2). A single dot product block in Figure 5.13 is implemented as a pipelined multiply-accumulate unit in order to increase the performance of the system.

Equation (5.3) expresses the hemispherical back-projection and coordinate system change from Cartesian to spherical. The L^2 norm sub-block in Figure 5.13 consists of two consecutive square root calculations. The square root module implements a CORDIC algorithm in the vectoring mode, which calculates the L^2 norm of its two inputs, *i.e.* $\sqrt{a^2 + b^2}$. The dividers for coordinate normalization are implemented using the iterative fast Anderson algorithm [68]. Transformation of the coordinate system requires evaluation of the inverse trigonometrical functions arctan and arccos. The spherical angles (θ, ϕ) are calculated by applying the CORDIC algorithm to the Cartesian coordinates, as illustrated in Figure 5.13.

The Angle Generation block provides information on the desired pixel grid. The error e from (5.4) is evaluated by the identical square root module used for the previous L^2 norm calculations. The error is compared to the pre-calculated ϵ, which is calculated by MicroBlaze using the camera calibration data. Output of the comparator serves as the output enable signal for a set of output registers, and as a write enable signal for the Camera Arbiter in Figure 5.7.

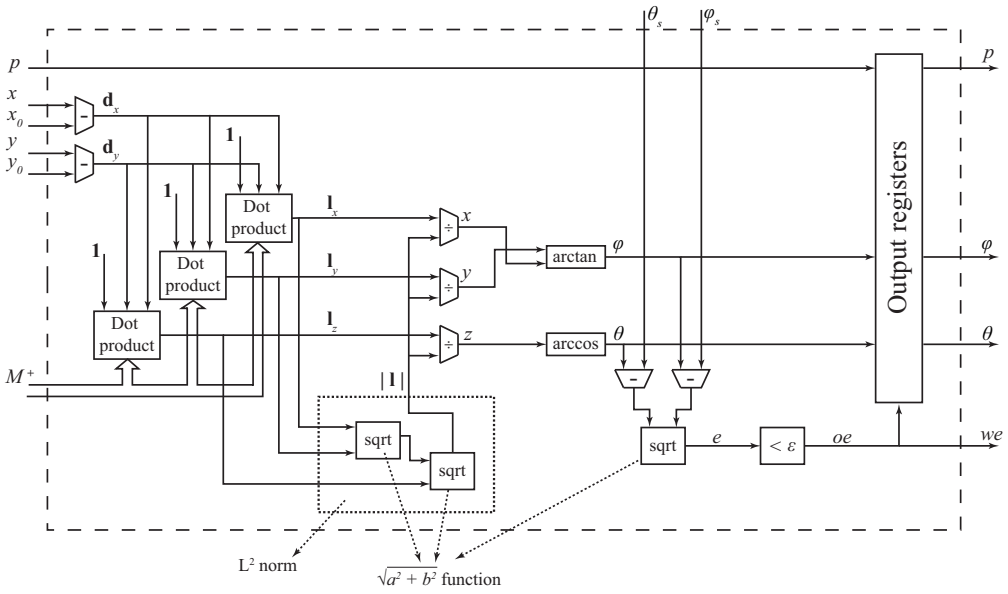


Figure 5.13: Internal architecture of Forward Homography Estimator. The presented hardware evaluates expressions (5.2) – (5.4) using pipelined architecture. Pipeline registers are not shown for better visibility. The sub-blocks for square root and trigonometric functions evaluation utilize the CORDIC algorithm, whereas the fast Anderson algorithm [68] is used for implementation of the dividers.

The FHE is a fully pipelined block. Each computation is followed by a register to shorten the critical path and increase the maximum frequency. Furthermore, each sub-block, *e.g.* dot product, square root, and trigonometric functions, is also pipelined providing a very fast operation. The pipeline registers are not shown in Figure 5.13 for clarity reasons.

5.4.5 Image Pyramids

Whether FHE is used or not, the Image Processing Unit operates identically. The Pixel Position module generates (x, y) coordinates of the pixel in $\boldsymbol{\omega}$ direction and fetches it from memory. *GigaEye II* system implements distributed architecture of MBB, and the core processing part is generation of the multi-resolution pyramids, as shown in Figure 5.14.

Apart from requesting pixels from the external memory, the Pixel Position informs the Image Pyramids block about the camera index, weight, and validity of the requested pixel. These three signals form a pixel descriptor, which is stored in the FIFO while the pixel is being read from the external DRAM. The weight can either be 1 if the camera is the best observing one for the selected pixel, or 0 otherwise. These weights correspond to the lowest level of the image LP, *i.e.* the high-frequency content as explained in Section 3.6. The *valid* signal indicates if the pixel is in the FOV of the camera. If the pixel is not in the FOV of the camera, the value 0 (black pixel) is sent to the corresponding FIFO instead of the pixel value read from DRAM.

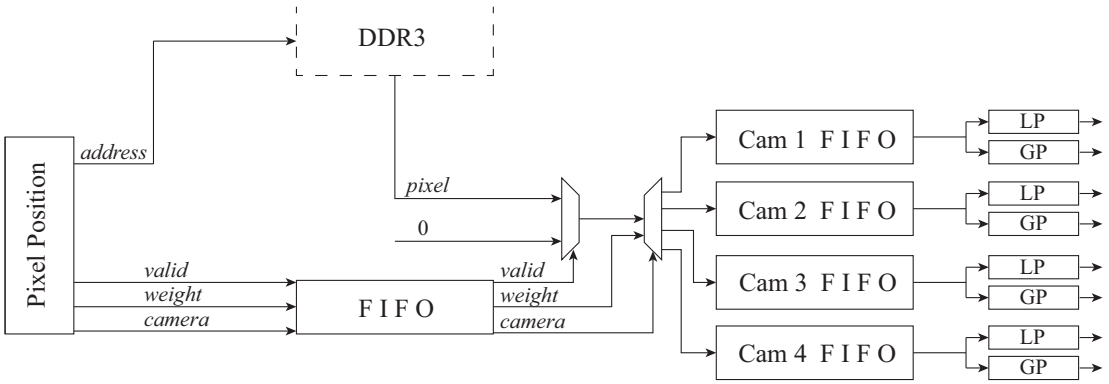


Figure 5.14: The block diagram of the Image Pyramids processing block. The block interfaces with both the Pixel Position block and the external memory. The block demultiplexes the pixel data with the camera index as a select signal, and sends it to the appropriate LP and GP decomposition circuit.

The pixel value and the weight are demultiplexed using the camera index as the select signal, and stored in one of the four FIFOs corresponding to the cameras in the system. The purpose of these FIFOs is to synchronize LP and GP decomposition. The FIFOs do not output pixels to the LP and the GP blocks until all four FIFOs have at least one stored pixel. This synchronization guarantees that image pyramids are created at the same time and the following blending block can safely implement weight multiplication.

The LP decomposition follows the algorithm illustrated in Figure 3.12 [74]. The same principle is applied to the GP decomposition, but without the downsampling and interpolation with **G(z)**. Both image quality and timing performance are dependent on the filters **H(z)** and **G(z)**. The 2D FIR filters are often used in FPGA and ASIC designs due to their inherent stability and simplicity of design in digital systems. The implementation of 2D FIR filters can be either separable or non-separable. The non-separable (direct) implementation consists of a 2D convolution of the filter matrix with the image. For $N \times M$ image resolution and $K \times K$ filter matrix size, the computational complexity of such filtering is $\mathcal{O}(MNK^2)$. The hardware design requires K^2 multipliers and a complex input buffer structure for larger filter sizes.

Oppositely, separable filters require less multipliers and adders compared to the direct implementation. However, the traditional separable implementation based on software algorithms is very resource-demanding and quite inefficient. Such computation is mathematically expressed as:

$$x' = (x * h_r)^T * h_c \tag{5.5}$$

where x and x' are the original and the filtered image, respectively, and h_r and h_c are row and column 1D filters. Operation denoted with $*$ represents a 1D convolution. Without loss of generality, in the rest of the thesis we will consider symmetric 2D filters, *i.e.* $h = h_r = h_c$.

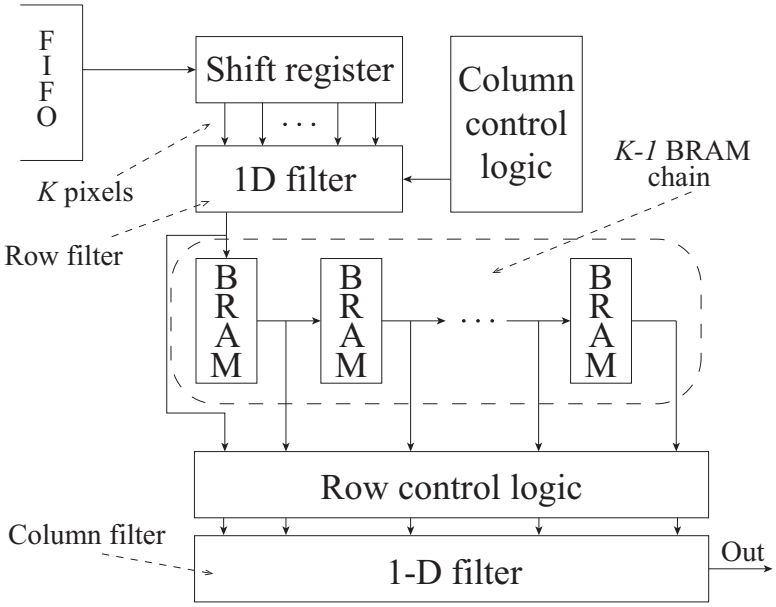


Figure 5.15: Internal architecture of the 2D separable filter. Both analysis and synthesis filters are implemented using the same architecture, with a slight difference in the control logic blocks.

The main issue of this implementation is the transposition block. Even though the complexity of $\mathcal{O}(MNK)$ is lower compared to the direct filtering, it requires more memory, as the whole intermediate image result is buffered. The buffering is obligatory due to reordering (transposing) of the pixels before the second 1D filter is applied, which increases the system latency by $N \times M$.

The internal architecture of the analysis and synthesis filters is shown in Figure 5.15. The goals of this design are real-time performance, and reduction of required hardware in the transposition block. Real-time performance is achieved by reducing the critical path delay using pipeline architecture, *i.e.* result of each arithmetic operation in the algorithm is followed by a register. Hence, the critical path is reduced to the length of the longest path in a single arithmetic block. Furthermore, the proposed design includes data sharing, which reduces number of memory read requests, increases performance and reduces hardware complexity.

Analysis filter

The analysis filter shown in Figure 5.15 operates as follows. The pixels that are ordered row-wise are read from one of the camera FIFOs. K pixels are buffered in a shift register, where K is the length of the used 1D filter. The pixels in the register are shifted with the arrival of each new pixel. All K pixels are available at the output and they are used by 1D row filter.

The row filter provides horizontally filtered pixels at its output. In standard separable filter implementations, these filtered pixels are stored in memory, transposed and filtered again.

However, using the proposed architecture, we avoid storing and transposing the full frame. The intermediate memory is replaced by a chain of $K-1$ line buffers, which are implemented as BlockRAMs in the FPGA.

Furthermore, not all filtered pixels are needed in the subsequent stages, because of the downsampling in the LP algorithm. Hence, we introduce two new blocks, named *Column control logic* and *Row control logic* in Figure 5.15. Since the filtered image will be downsampled, we distribute the downsampling operation into row and column procedures, and embed it in the hardware filter. When one pixel is filtered by the row filter, the *Column control logic* disables the filtering of the next pixel, *i.e.* pixel positioned in the next column. After skipping one pixel, the control logic again enables the filter. This principle is repeated for all pixels in the image, and it corresponds to the horizontal downsampling by two.

The pixels belonging to the same row are buffered in the same BlockRAM, and only $K-1$ half-rows are stored in this chain thanks to the control logic. Whenever a new filtered pixel arrives, it is stored in the first BlockRAM at the location addressed by the pixel's column in the frame. Since the utilized BlockRAMs behave as a dual-port memory, the second port is used for reading the pixel from the same memory location, *i.e.* the pixel in the same column from the previous row. The read pixel is then stored in the following BlockRAM in the chain. Hence, this BlockRAM chain can also be regarded as a set of stacked shift registers.

The outputs of $K-1$ BlockRAMs and the output of the row filter form a set of K vertically neighboring pixels. Hence, the transposition is no longer required, as the pixels are available in the appropriate order. Similar to *Column control logic*, *Row control logic* block disables filtering of every second row in the column filter. It is important to note that even when column filtering is disabled, shifting of pixels between BlockRAMs is enabled. This is obligatory due to the fact that one source pixel contributes to $(K+1)/2$ filtered pixels in a single column.

The pixels allowed through the *Row control logic* are filtered using the second 1D filter (column filter in Figure 5.15) and streamed out to the rest of the processing system. The outputs are sorted in the same order as the original input, *i.e.* in the row-wise order.

Synthesis filter

Opposite to the analysis filter **H(z)** that downsamples the image, the synthesis filter **G(z)** upsamples it. A property of the upsampling operation is that output data rate of the filter is higher than the input data rate. We implement a time-multiplex system to resolve this issue, under the safe assumption that the blending operation does not increase the data rate.

The synthesis filter is implemented using the same top-level architecture as the analysis filter. The main difference is in the control logic blocks of the synthesis filter, since it multiplexes the input pixels with the upsampled zero-valued pixels. When the filter receives a pixel from i^{th} column, the *Column control logic* allows the row filter to output pixel from $i-(K-1)/2$ column. In the following clock cycle, the logic will enable the filter to output the $i-(K+1)/2$

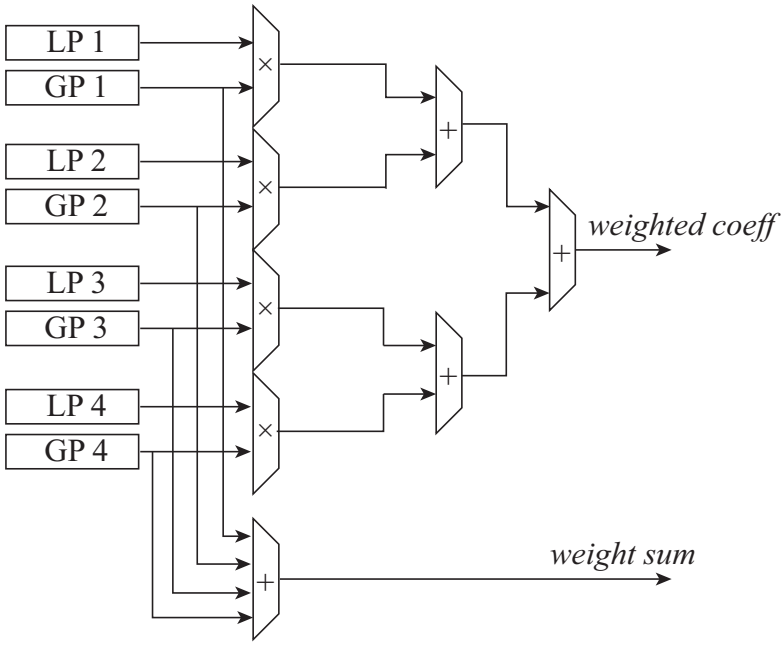


Figure 5.16: Block diagram of the Image Blending module in the cluster FPGA of *GigaEye II* .

column. The insertion is allowed because of two reasons: (1) the corresponding input pixel for the second output pixel is zero, and (2) the assumption that input data rate is not faster than the output rate of the LP decomposition. Levels l_i, for $i = \{2, ..., L\}$ cannot provide pixels in each clock cycle, hence upsampling of the pyramid levels can be embedded in the filtering operation. Level l_1 is the only level that can theoretically provide pixels every cycle, but its pixels are not being filtered by **G(z)** during the reconstruction (see Figure 3.12).

The line buffers store the upsampled rows. The *Row control logic* operates on the same principle as *Column control logic* with the exception that it inserts the row pixels. When the column filter provides a pixel from i^{th} row, the *Row control logic* enables the column filter to output the pixel from $i-(K-1)/2$ row. In the following clock cycle, a pixel from $i-(K+1)/2$ row will be calculated.

The outputs of the synthesis filter are the levels of LP, and they are the inputs of the Image Blending block.

5.4.6 Image Blending

The hardware implementation of the Image blending block is depicted in Figure 5.16. Similarly to *Panoptic* , the Image Blending module in *GigaEye II* conducts the final step of the partial panorama construction. The module receives frames from four cameras decomposed into LP, and their corresponding weight GP. The coefficients from image LP are multiplied by the weights from the GP. Furthermore, weights from all four GPs are summed to obtain the

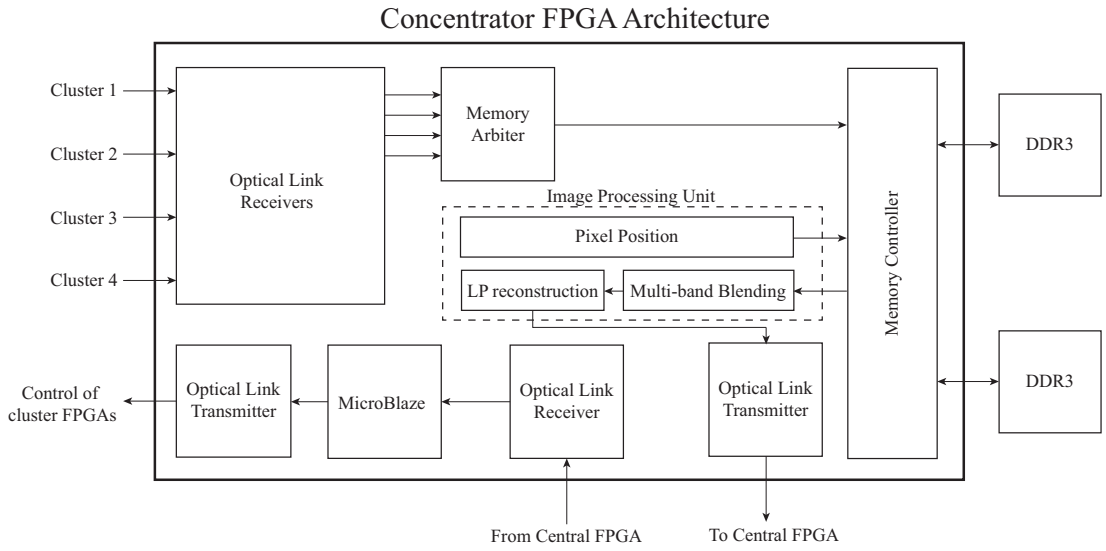


Figure 5.17: Top-level architecture of the *GigaEye II* concentrator FPGA.

normalizing factor, as in (3.12). However, the weighted LP coefficients are not normalized at this moment, since they may final blending is performed in the concentrator FPGA. As the blending result in the cluster FPGA, the weighted LP coefficients and the sum of all corresponding weights are sent to the concentrator FPGA via the optical link transmitter.

5.5 Concentrator Processing Board

The concentrator processing board collects data from the four cluster boards. The Memory Arbiter block behaves in the identical manner as the arbiter in the cluster FPGA, *i.e.* four input channels are served in the round-robin pattern. The DDR3 memory mapping is also identical to the one in the cluster FPGA, with each camera stream replaced by the stream coming from the cluster FPGA. The main difference between the cluster and the concentrator FPGA design is in the Image Processing Unit that is simplified. There is no need for the angle or $\boldsymbol{\omega}$ generation, since there is no direct access to the camera frames. The pixels are already arranged in a sequential order. Thus, a counter is instantiated in the Pixel Position block, whose value is linearly mapped to a memory address.

Recall that the cluster boards calculate and send both the weighted LP coefficient and the sum of corresponding weights. These values are stored in DRAM of the concentrator board, and read by the Image Processing Unit. The MBB block sums all weighted LP coefficients, sums all weights from GP, and normalizes the weighted sum, as shown in Figure 5.18. The resulting value is the blended LP coefficient of the final panorama.

The final panoramic frame is reconstructed from the blended LP coefficients using the chain of interpolation filters, as shown in Figure 3.12. The reconstructed panorama is transmitted to the central board via optical link.

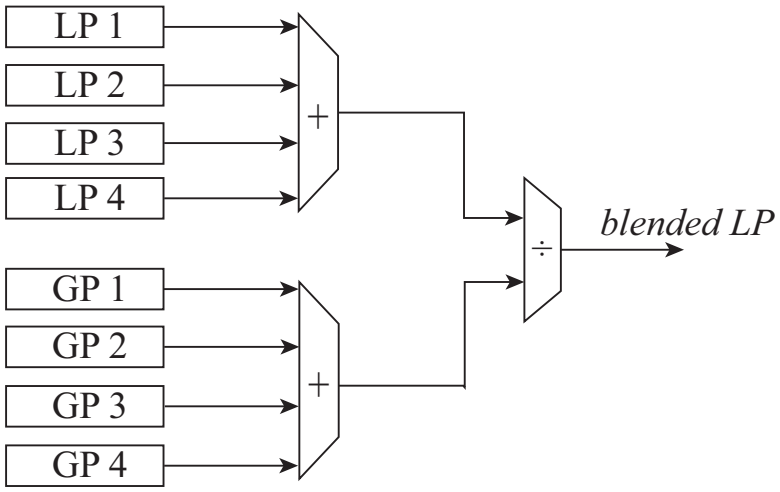


Figure 5.18: Block diagram of the Image Blending module in the concentrator FPGA of *GigaEye II* .

Apart from the data path, the concentrator boards also includes the system control path. The commands from the central FPGA are decoded, and interrupt is raised in the MicroBlaze. The interrupt routine is serviced, and the command is forwarded to the cluster FPGA if needed.

5.6 Central Processing Board

The top-level architecture of the central FPGA is shown in Figure 5.19. The central FPGA is connected to the concentrator board via 10 Gb/s optical link. The Optical Link Receiver receives the already constructed panoramic frame, generates the valid memory address, and stores it into a single DDR3 module on the board. The memory dedicated to video streaming is divided into two pages, where one frame occupies one page. While the current frame is being written into the memory, the previous one is read by the display controller. Thus, it is not possible to change the contents of the frame being displayed, which leads to possible image "tearing" in the presence of fast moving objects.

The display controller consists of five instantiated HDMI controllers, allowing five independent outputs. The controllers have direct memory access, and read the already stored full frames. Since the VC707 board has only one HDMI output, two FMC extension boards are used to provide additional display capabilities. These extension boards are shown in Figure 5.20.

The central FPGA is also used as an interface towards the user, *i.e.* a client PC. It has both USB2 and UART connectors. Since there is no high data rate transfer between the PC and the central FPGA, only UART link is used to send commands from the PC. The user can directly set any software accessible register in the central FPGA through MicroBlaze interrupt routine. Furthermore, it can indirectly set any concentrator and cluster FPGA register, as well as any camera register, since the commands are propagated via the Optical Link Transmitter.

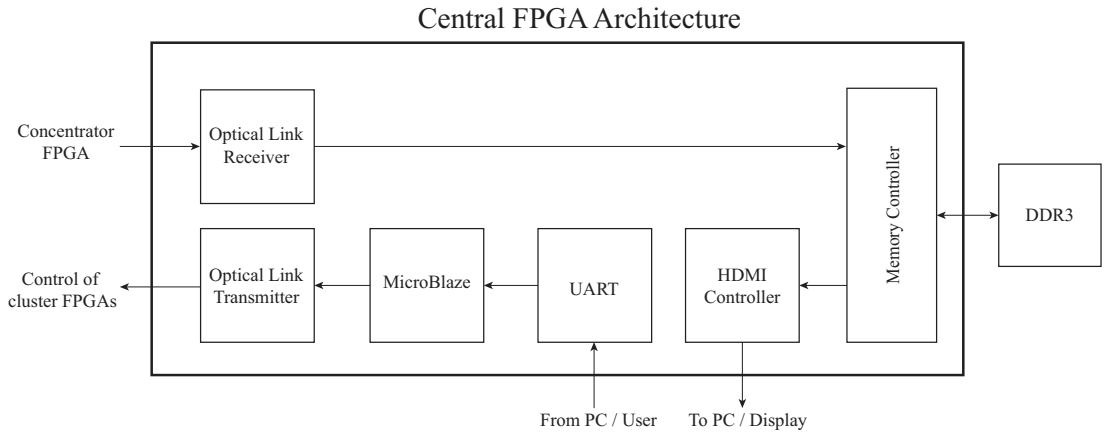


Figure 5.19: Top-level architecture of the *GigaEye II* central FPGA.

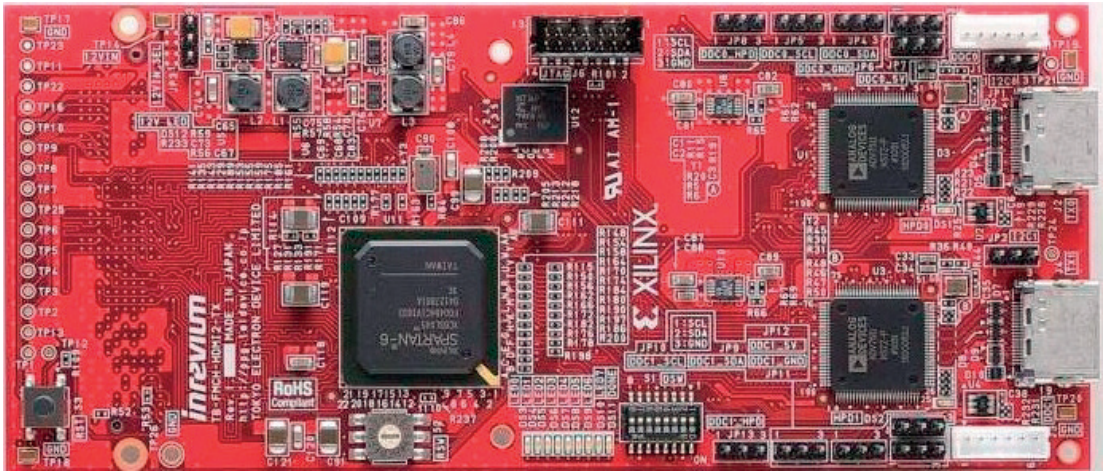


Figure 5.20: FMC extension board providing two additional HDMI outputs.

5.7 Experimental Results of the *GigaEye II* System

The camera mount ring for *GigaEye II* with diameter of $2r = 50\ cm$ is built using a 3D printer. The camera ring can currently accommodate thirty-two CMOSIS CMV20000 cameras, arranged on two floors. The hemisphere populated with cameras is positioned on top of an aluminum rack that holds six processing FPGA boards and the power supply. The whole structure is shown in Figure 5.21. The camera modules are connected to the FMC interface PCB using the high-speed SAMTEC Q Rate differential cables, and they are operated at 30 fps.

The architecture presented in this chapter was developed in VHDL for the target FPGAs. Similarly to *Panoptic* , the developed firmware conducts all mathematical processing using 16-bit fixed-point precision. The firmware was targeted and successfully tested for operation at 233 MHz f_{clk} frequency on the cluster and concentrator FPGAs, and 200 MHz on the central FPGA. These frequencies are chosen since they are the frequencies of the on-board oscillators. The resource utilization of each board separately is shown in Table 5.2.

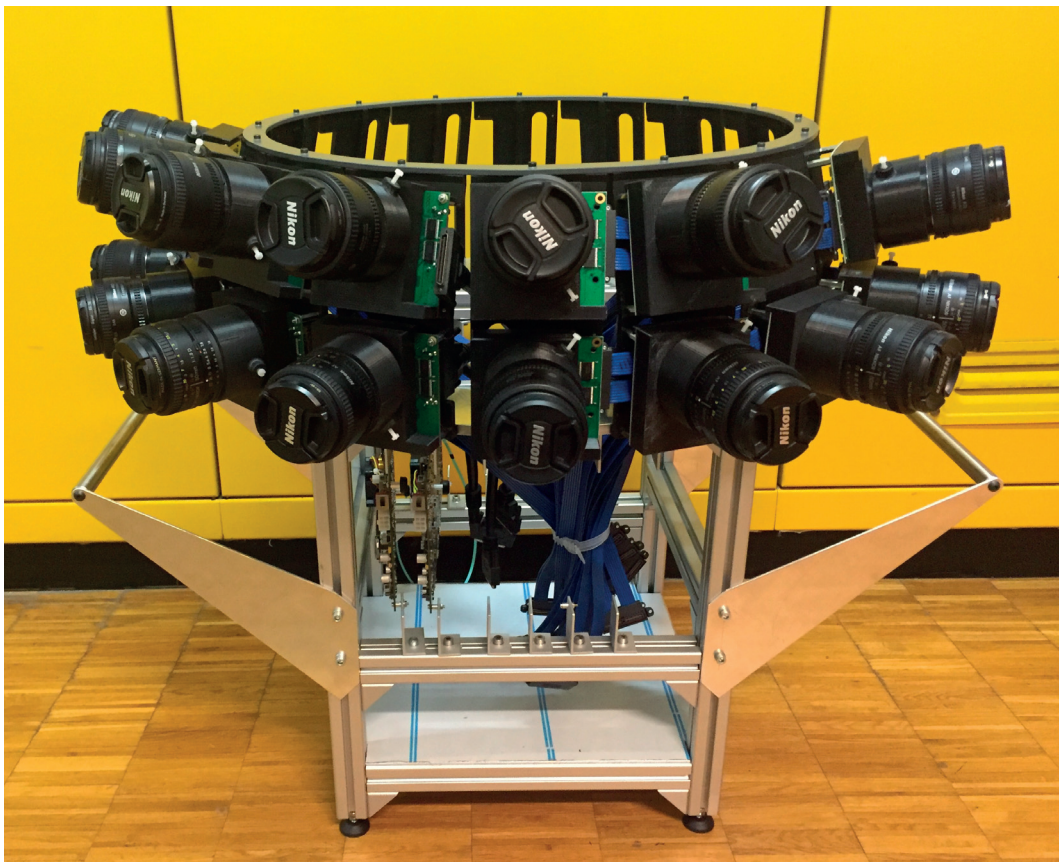


Figure 5.21: The mounted cameras on the *GigaEye II* camera system.

Due to the limits of the modern-day displays, the final real-time output resolution is 1080p30, *i.e.* 1920 × 1080 pixel at 30 fps. However, thanks to 20 Mpixels cameras and the adaptable reconctruction algorithm, it is again possible to reconstruct a narrower FOV while keeping the same output resolution, providing more details in that area.

Table 5.2: *GigaEye II* Virtex-7 FPGA resource utilization summary.

Resource	**Cluster**	**Concentrator**	**Available**	**Central**	**Available**
Flip-flop	118246	24902	866400	21403	607200
LUT	79575	25614	433200	23423	303600
BlockRAM	386	112	1470	81	1030
DSP	131	8	3600	6	2800
MMCM	7	4	20	4	14
PLL	2	4	20	1	14
BUFG	18	12	32	12	32

5.8 Conclusion

In this chapter, a real-time high-resolution multi-camera system *GigaEye II* is presented. *GigaEye II* is implemented using the distributed processing approach. The full design was detailed, including the camera choice and specifications, the PCB design, and the multi-board real-time hardware implementation of the omnidirectional view construction. The hardware resource utilization of all processing FPGAs in the system is also provided.

We presented one possible solution to the problem of designing a high-resolution real-time multi-camera system. The distributed approach for implementing real-time applications in multi-camera systems is very efficient in terms of processing power and speed, but not easy to design and synchronize. Thanks to the workload distribution and parallel implementations, this system achieves the high resolution and the high frame rate operation. The presented embedded architecture with the true distributed workload, *i.e.* each board reconstructing only a partial FOV is novel in the field, and provide numerous application possibilities.

Finally, the system-level architecture of *GigaEye II* with the cluster and the concentrator boards allows straightforward scalability of the system. Addition of new four-camera clusters does not require any change in system's architecture, or the used algorithm thanks to fully distributed reconstruction. Hence, *GigaEye II* can easily reach the Gigapixel resolutions with enough number of installed cameras.

6 Computational Imaging Applications

The previous two chapters introduced a multi-camera system design and FPGA implementation for real-time omnidirectional video construction in both low and high resolution. This chapter will present three potential applications of multi-camera systems such as *Panoptic* or *GigaEye II* . The first application that will be presented is the multiple regions of interest view using *GigaEye II* system. The second application is the HDR imaging implemented on a platform Panoptic Media Platform [31], but it is easily portable to any other multi-camera system with large FOV overlap between the cameras. Finally, the third application is an object detection and tracking system implemented on a heterogeneous platform, *i.e.* a combination of the FPGA-based omnidirectional video generation and a GPU running the object detection and tracking software.

6.1 Multiple Regions of Interest

GigaEye II camera system records approximately 320 Mpixels each frame, but only 2 Mpixels can be displayed on a standard Full HD screen (or 8 Mpixels on the latest 4K screens). Due to the limitation of displays, all the details cannot be seen in the real-time panoramic stream. Thus, *GigaEye II* has the capability to display selected regions of interest on different screens using the FMC extension board shown in Figure 5.20.

Top part of Figure 6.1 shows a snapshot from the panoramic stream, and three regions of interest marked with red rectangles. The regions are selected by the user, and the information about their position is transmitted to the central FPGA board. The central unit determines which individual camera in the system observes the selected region, and indicates to the appropriate cluster FPGA that a video stream from that camera is required. When the cluster FPGA receives this interrupt, it starts sending a Full HD frame centered around the desired point in addition to constructing the partial panoramic frame. This single camera Full HD frame is propagated through the concentrator FPGA to the central unit, and displayed using one of the HDMI controllers.

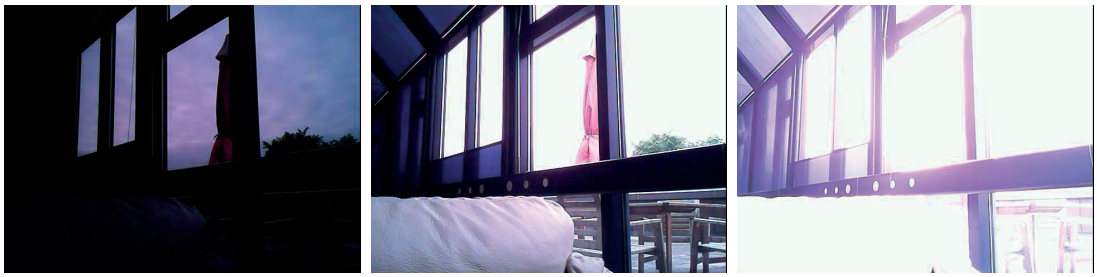
Figure 6.1: A high-resolution panoramic frame with three selected regions of interest shown in full resolution.

(a)

(b)

Figure 6.2: (a) An example of a multi-display setup used to display data from *GigaEye II* , (b) an illustration of how the user can visualize the multiple regions of interest in parallel with the full omnidirectional view.

This capability offers a multi-display setup as shown in Figure 6.2. This application is targeted at surveillance systems, where a human operator is controlling the camera and observing the omnidirectional view. Any region in the main view can be selected for a more detailed, high-resolution view, and *GigaEye II* instantaneously provides an additional single camera stream around the selected region.



(a) (b) (c)

Figure 6.3: A subset of images taken for recovering the camera response curve. The images are taken with (a) short, (b) medium and (c) long exposure time.

6.2 High Dynamic Range Imaging

6.2.1 Introduction

Dynamic range in the digitally acquired images is defined as the ratio between the brightest and the darkest pixel in the image. Most modern cameras cannot capture sufficiently wide dynamic range to truthfully represent radiance of the natural scenes, which may contain several orders of magnitude from light to dark regions. This results in underexposed or overexposed regions in the taken image and the lack of local contrast. Figure 6.3 shows three shots taken under different exposure settings of a camera. The underexposed and overexposed images show fine details in very bright and very dark areas, respectively. These details cannot be observed in the moderately exposed image.

HDR imaging technique was introduced to increase dynamic range of the captured images. HDR imaging is used in many applications, such as remote sensing [75], biomedical imaging [76] and photography [77], thanks to the improved visibility and accurate detail representation in both dark and bright areas.

HDR imaging relies on encoding images with higher precision than standard 24-bit RGB. The most common method of obtaining HDR images is called exposure bracketing and it includes taking several low dynamic range (LDR) images, all under different exposures [78]. Debevec and Malik [79] developed an algorithm for creating wide range radiance maps from multiple LDR images. The algorithm included obtaining camera response curve, creation of HDR radiance map and storage in RGBE format [80]. Other approaches based on a weighted average of differently exposed images were proposed in [81, 82, 83], with differently calculated weights. State-of-the-art algorithms for radiance map construction include camera noise model and optimization of the noise variance as the objective function [84, 85].

An exposure bracketed sequence can also be fused into the HDR image without the radiance map calculation. Exposure fusion method [86] is a pipelined fusion process where LDR images are combined based on saturation and contrast quality metrics. Thanks to direct image fusion,

the exposure fusion is not a resource-demanding algorithm as there is no HDR radiance map to be stored, which significantly reduces the memory requirement. A similar principle is used for contrast enhancement using a single LDR image [87]. An alternate option to exposure bracketing is to use an adjustable camera response curve sensor, such as LinLog [88].

Besides capturing the natural scenes, another problem occurs when displaying them. The modern displays are limited to the low dynamic range, which causes inadequate representation of even standard LDR images. In order to avoid such problems, a tone mapping operation is introduced to map the real pixel values to the ones adapted to the display device. The purpose of tone mapping is to compress the full dynamic range in the HDR image, while preserving natural features of the scene.

Tone mapping operators can be divided into two main groups named global and local operators. Global operators are spatially invariant because they apply the same transformation to each pixel in the image. These algorithms usually have low complexity and high computational speed. However, such algorithms have problems preserving the local contrast in the images where the luminance is uniformly occupying the full dynamic range. The first complex global techniques were based on human visual system (HVS) model and subjective experiments [89, 90]. The latest global techniques are based on adaptive mapping. Drago et al. [91] introduced an adaptive logarithmic mapping which applies different mapping curves based on pixel luminosity. The curves vary from $\log_2$ for the darkest pixels, to $\log_{10}$ for the brightest. Similarly, Mantiuk et al. [92] have recently developed a tone mapping algorithm adaptive to the display device.

Opposite to the global operators, local operators are more flexible and adaptable to the image content, which may drastically improve local contrast in regions of interest. Since they differently operate on different regions of the image, they are computationally more expensive and resource-demanding. Reinhard et al. [93] introduced a local adaptation of a global logarithmic mapping. The adaptation was inspired by photographic film development in order to avoid halo artifacts. Fattal et al. [94] proposed an operator in gradient domain which was computationally more efficient than other local operators. Nevertheless, both Reinhard and Fattal operators are very resource-demanding for large images, since they require a Gaussian pyramid decomposition and a Poisson equation solver, respectively. Durand and Dorsey [95] presented a fast bilateral filtering where high contrast areas are preserved in the lower spatial frequencies. However, the main disadvantage of this method is the significantly lower overall brightness.

Obtaining and reproducing the HDR video is a difficult challenge due to various issues. Majority of the techniques use exposure bracketed frames from a single camera, which results in high motion blur among frames. Furthermore, using frames from the same camera inherently lowers the effective frame rate of the system, independently of the tone mapping process. The display frame rate is further influenced by both the HDR imaging technique and the processing system. Majority of the systems are based on CPU or GPU. Even though GPUs are

targeted to process large amount of data in parallel, they often fail to meet the tight real-time timing constraints.

This Section presents simultaneous real-time HDR video construction and rendering using a multi-camera system [31]. The key idea is to use a multi-camera setup to create a composite frame, where cameras with the overlapping FOV are set to different exposure times. Such system reduces the motion blur, as there is no inter-frame gap time (which can be several hundreds milliseconds in the standard HDR cameras). Additionally, the frames are captured at the same moment by all cameras, which reduces the intra-frame motion of the scene objects to the difference interval of cameras' exposure times.

6.2.2 Related Work

Exposure bracketing using a single video camera is the most widely used method for HDR video construction. Kang et al. [96] proposed a method of creating a video from an image sequence captured while rapidly alternating the exposure of each frame. Kalantari et al. [97] apply the identical principle and use the patch-based synthesis to deal with the fast movements in the scene. The HDR construction in both cases is realized in post-processing and does not have the real-time processing capability. Gupta et al. [98] recently proposed a way of creating HDR video using Fibonacci exposure bracketing. In this work they adapted a machine vision camera Miro M310 to quickly change exposures and thus reduce the inter-frame delay. However, this system still requires significantly long time to acquire a sequence of frames with desired exposures. Another approach is to use a complex camera with beam splitters [99]. A similar setup is also used in the work of Kronander et al. [100]. This spatially adaptive HDR reconstruction algorithm fits local polynomial approximation to the raw sensor data. However, the algorithm requires intensive processing to recover and display the HDR video.

Exposure bracketing can also be used in multi-camera or multi-view setups. Ramachandra et al. [101] proposed a method for HDR deblurring using already captured multi-view videos with different exposure times. Portz et al. [102] presented a high-speed HDR video using random per-pixel exposure times. This approach is a true on-focal-plane method which still needs to be implemented on a sensor chip.

High frame rate HDR imaging is a challenging problem, even with state-of-the-art processing units. Thus, many attempts have been made to develop a dedicated hardware processing system for this purpose. Hassan and Carletta [103, 104] proposed an FPGA architecture for Reinhard [93] and Fattal [94] local operators. Even though the proposed implementations concern only the tone mapping operator, the designs require a lot of resources. This originates from the Gaussian pyramid and LUT implementation of the logarithm function [103] and a local Poisson solver [104]. Another FPGA system was implemented by Lapray et al. [105, 106]. They presented several full imaging systems on Virtex-5 FPGA platform as a processing core. The system uses a special HDR monochrome image sensor providing a 10-bit data output.

Apart from FPGA systems, GPU implementations of full HDR systems are also available. Akyüz [107] presented a comparison of CPU and GPU processing pipelines for already acquired bracketed sequences. Furthermore, real-time GPU implementations of different local tone mapping operators can be found in [108, 109].

6.2.3 Camera Prototype

A custom-made FPGA platform is used for the practice of the real-time omnidirectional video system [31]. The assembled prototype is shown in Figure 6.4a. The designed prototype is an FPGA-based processing platform, which includes eight Xilinx XC5VLX110 Virtex5 FPGAs. One FPGA is targeted for the implementation of the central/master unit and the other seven are slaves used for camera interfacing and local processing on the camera level.

The central FPGA hosts the central processing unit of the system. It is designed to be in charge of system initialization, timing synchronization among the FPGAs, inter-FPGA communication control, video display, and Gigabit Ethernet and USB 2.0 links to a PC. Role of the slave FPGA is to create a partial composite frame and send it to the central unit for display. Each slave FPGA is capable of interfacing seven imagers and seven 2 MB SRAM modules, due to limited number of available user I/O pins. Hence, each imager has a dedicated memory storage, which allows the processing unit to simultaneously access pixels from several cameras.

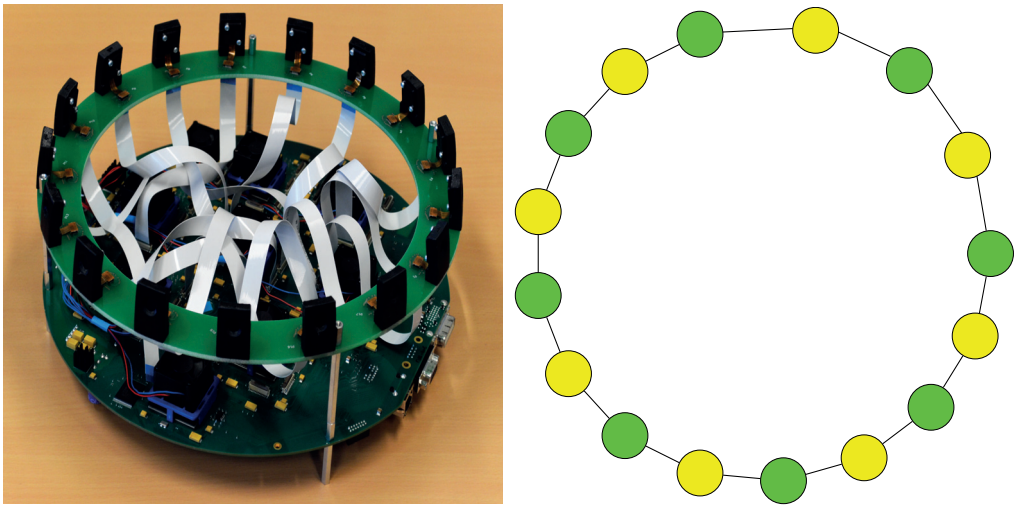
The board is capable of interfacing maximum forty-nine cameras to achieve the full hemispherical view. For the purpose of HDR video application, sixteen cameras are placed on a circular PCB ring. The PCB ring in Figure 6.4a is $2r = 30$ cm in diameter. Low-cost cell-phone VGA cameras, with the minimum FOV of 46°, are placed and operated at 25 fps. The graph representation of the camera connections is given in Figure 6.4b. Each camera is able to communicate, *i.e.* share pixel data, with at most two neighboring cameras. Thanks to the inter-FPGA connections, cameras are able to obtain information from a neighboring camera, even if they are not connected to the same FPGA.

6.2.4 HDR Video

The pixel streams coming from the cameras are processed in real-time; hence, HDR video is created as a stack of HDR frames in time domain. Construction of each frame can be divided into two independent processes: (1) construction of HDR composite frame, and (2) tone mapping the composite frame to achieve realistic rendering.

HDR Composite Frame

The circular arrangement of the cameras on this prototype allows us to generate the panorama using the Gaussian blending method detailed in Section 3.5, simplified to a two-dimensional camera arrangement case. To be able to reproduce the HDR image, the cameras are color



(a) (b)

Figure 6.4: (a) The built prototype with processing board at the bottom and the installed camera PCB ring. The diameter of the system is $2r = 30$ cm; (b) The graph representation of the camera arrangement. The yellow and green circles represent the cameras with long and short exposure times, respectively. The links between cameras are drawn and each camera can communicate (share pixel values) only with the differently exposed neighbors.

calibrated. The camera's response curve is recovered using a set of shots of the same scene with different exposure settings. Three out of twelve taken images are shown in Figure 6.3. The response curve is recovered by applying the algorithm proposed by Debevec and Malik [79]. Only one camera is color calibrated, as it is assumed that the response curve is identical for all installed cameras.

FOVs of the cameras overlap such that each point in space is observed by at least two cameras [110]. We exploit this property and set the camera exposures to different values. During the camera initialization phase, all cameras are set to the auto-exposure mode. The camera with the longest exposure time, *i.e.* the one observing a dark region, is taken as a reference. In the following step, half of the cameras are set to the reference exposure t_{ref}, while other half is set to $t_{ref}/4$, such that two cameras with overlapping FOVs have different exposure times. The resulting diagram is shown in Figure 6.4b, where the yellow and green circles represent cameras with long and short exposure times, respectively.

The calibration data provides yaw, pitch and roll data for each camera. We are able to determine tessellation of the hemispherical projection surface according to the influence of the cameras, using these Euler angles and the focal length of the camera [54]. Each 3D region in the obtained tesselation denotes a solid angle in which the observed camera has dominant influence, whereas boundaries of these regions are lines of identical influence of two cameras. This tesselation is called the Voronoi diagram [111]. The most influential camera within a single tile is called the *principal camera*.

As the calibration parameters are known, the composite image is constructed by projection of the camera frames onto the hemispherical surface. In order to obtain the HDR radiance map, the color calibration data should be included as suggested in [79]. The pixel values C_i are expressed as:

$$\ln C_i = \frac{\sum_j w(I_{j,i}) \cdot \left(g(I_{j,i}) - \ln t_{ref,j}\right)}{\sum_j w(I_{j,i})} \tag{6.1}$$

$$w(I_{j,i}) = \begin{cases} I_{j,i} - I_{j,min} & \text{, if } I_{j,i} \leq \frac{1}{2}(I_{j,min} + I_{j,max}) \\ I_{j,max} - I_{j,i} & \text{, otherwise} \end{cases} \tag{6.2}$$

where j is the camera index, i is the pixel position, C is the composite image, $I_{j,i}$ represents a set of pixels from contributing cameras, g is the camera response function, and $I_{j,min}$ and $I_{j,max}$ are minimum and maximum pixel intensities in the observed camera frame. The camera response function is recovered using the approach by Debevec [79], and it is shown in Fig 6.5.

The nature of the HDR imaging is to recover the irradiance using sensors with different exposures. Hence, we constrain the expression (6.1) by evaluating it using only two contributing cameras with mutually different exposures. The second camera is referred to as the *secondary camera*.

The calculated piecewise linear weights $w(I_{j,i})$ are included in the Gaussian blending process as:

$$w'(I_{j,i}) = w(I_{j,i}) \cdot \frac{1}{r_j} \cdot e^{-\frac{d_i^2}{2\sigma_d^2}} \tag{6.3}$$

where the notation is kept identical to (6.1) and (6.2), with r_j as the distance of camera's projection from the observer, and d_i as the pixel distance from the frame center. High standard deviation σ_d increases region of influence of each camera; hence, relative influence of the *principal camera* is reduced. This results in a smoothly blended background and increased ghosting around edges of the objects in the scene. Thus, the standard deviation σ_d is empirically determined for the given camera setup in order to obtain the best image quality.

The result of applying equations (6.1) and (6.3) on the acquired data provides the composite HDR radiance map, which should be tone mapped for realistic display.

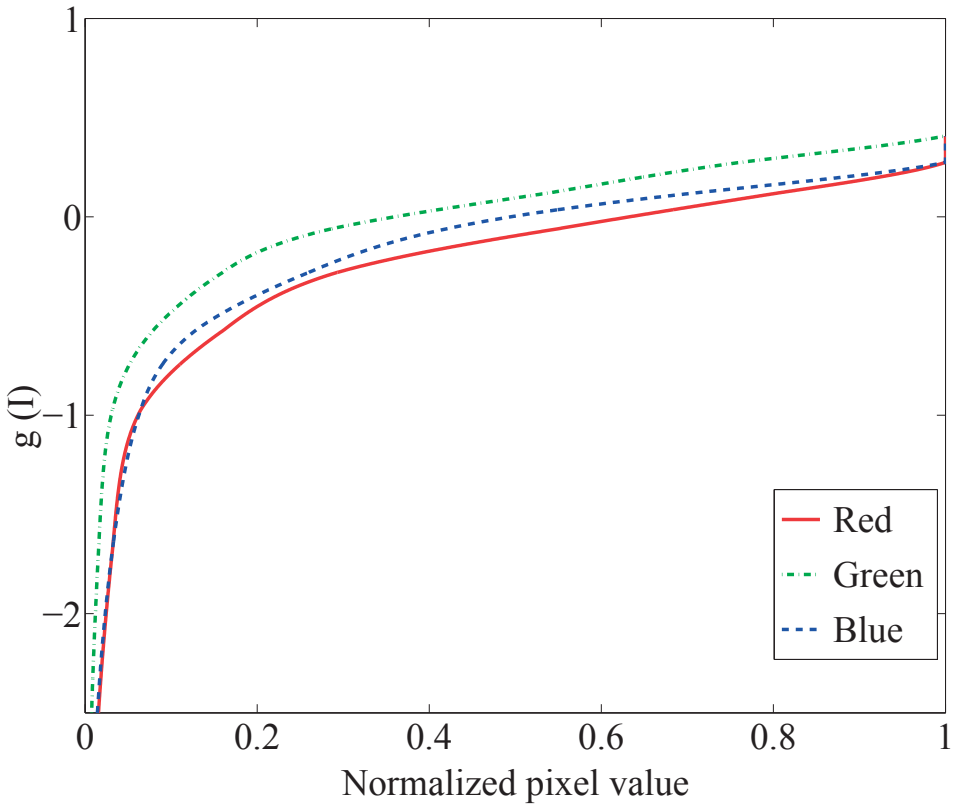


Figure 6.5: Recovered response function $g(I)$ of a single camera. Three curves correspond to red, green, and blue pixels, as shown in the legend.

Tone Mapping

Yoshida et al. [112] made an extensive comparison of the tone mapping operators. The comparison was realized by human subjects grading several aspects of the constructed image, such as contrast, brightness, naturalness and detail reproduction. One of the best graded techniques in this review was the local operator by Drago et al. [91]. Therefore, this operator will be taken as a base for the development of an FPGA-suitable operator. Similar to majority of the global operators, this operator uses logarithmic mapping function expressed in (6.4), where displayed luminance L_d is derived from the ratio of world luminance L_w and its maximum L_{max}. The algorithm adapts the mapping function by changing the logarithm base t as a function of the bias parameter b, as shown in (6.5).

$$L_d = \frac{\log_t(L_w + 1)}{\log_{10}(L_{max} + 1)} \tag{6.4}$$

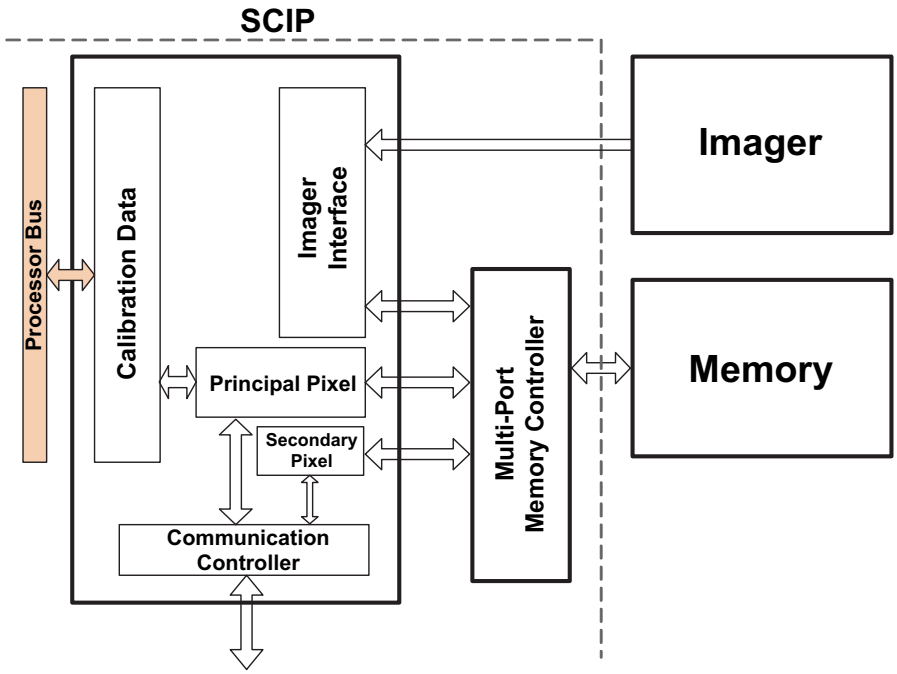


Figure 6.6: Internal blocks of the smart camera IP used in the slave FPGAs.

$$t(b) = 2 + 8 \cdot \left(\frac{L_w}{L_{max}} \right)^{\frac{\ln b}{\ln 0.5}} \tag{6.5}$$

Even though this mapping is created for interactive applications, its speed is very slow for video applications. The reported frame rate is below 10 fps, for 720 × 480 pixels image, without any approximations which decrease the image quality [91]. Calculation of the logarithm values is the most process-intensive part [113], whether the algorithm is implemented on CPU or GPU. We have derived an operator suitable for direct hardware implementation which reduces the calculation time.

Drago et al. [91] proposed changing logarithm base and calculating only natural and base-10 logarithms. However, fast logarithm calculations are very resource-demanding, because they require large pre-calculated LUTs. Hence, we approximate the logarithm of the form $\log(1+x)$ by the Chebishev polynomials of the first kind $T_i(x)$ [68]. This approximation needs only 6 integer coefficients to achieve 16-bit precision, which is enough for log-luminance representation in this camera. The Chebyshev approximation can be applied to both natural and base-10 logarithm by only changing the coefficients. The coefficients for the natural logarithm are denoted as $c_e(i)$, while $c_{10}(i)$ are for base-10 in (6.6).

According to [91], the best visually perceived results are obtained for the bias parameter $b \approx 0.85$. Fast calculation of generic power functions, *e.g.* the one required in (6.5), is not

possible. Hence, we fixed the parameter to $b = 0.84$, to relax the hardware implementation, without losing any image quality. The exponent is then 0.25, and the result can be evaluated by two consecutive calculations of the square root. The square root is also approximated by the Chebyshev polynomials. The expanded operator is expressed as:

$$L_d = \frac{\sum_{i=0}^{5} c_e(i)\, T_i\,(L_w)}{\sum_{i=0}^{5} c_{10}(i)\, T_i\,(L_{max}) \cdot \ln\left(2 + 8\left(\frac{L_w}{L_{max}}\right)^{\frac{1}{4}}\right)} \tag{6.6}$$

The natural logarithm term in the denominator cannot be precisely approximated by Chebyshev polynomials, due the arguments much higher than 1. A suitable approximation of the expression $\ln x$ is a fast convergence form of the Taylor series, which is expressed in (6.7). This expression needs only 3 non-zero coefficients to achieve a sufficient 16-bit precision, but the argument should be preconditioned as shown:

$$\ln x = 2 \sum_{k=1}^{3} \frac{1}{2k-1} \left(\frac{x-1}{x+1}\right)^{2k-1} \tag{6.7}$$

The equations (6.6)-(6.7) describe the new tone mapping operator suitable for hardware implementation. The set of required mathematical operations is reduced to only addition, multiplication and division, which are suitable for fast implementation.

6.2.5 FPGA Implementation

Local Processing

The processing platform consists of seven slave FPGAs used for local image processing, and each slave unit can be connected to seven cameras, due to I/O pin availability. Local processing is realized on the camera level, utilizing the custom-made Smart Camera Intellectual Property (SCIP) shown in Figure 6.6. SCIP is instantiated for each camera in the system, and it is in charge of creating a partial HDR composite within camera's FOV.

Responsibilities of each SCIP are three-folded: (1) Acquire pixels from the imager and store them in memory, (2) Evaluate the HDR pixel value where the selected camera is the principal camera, and (3) Provide pixel value to the principal camera, when the selected camera is the secondary camera. The Imager Interface in Figure 6.6 receives the pixel stream from the camera and stores in the memory. Calibration data block stores information about position of all cameras which are physically close to the observed camera. Thus, SCIP determines the local Voronoi tessellation, and calculates both principal and secondary weights for the camera. The distributed implementation of the algorithm from Section 6.2.4 is summarized in Algorithm 3.

Algorithm 3 Smart Camera Processing

1: calculate calibration data
2: calculate weights
3: **for all** principal pixels **do**
4: $p_m := read\_pixel\_from\_memory$
5: $p_{s,in} := request\_pixel\_from\_secondary\_camera$
6: $C := \frac{w'_m \cdot p_m}{w'_m + w'_s} + p_{s,in}$
7: send C to central unit
8: **end for**
9: **for all** secondary pixels **do**
10: wait for request from principal camera
11: $p_s := read\_pixel\_from\_memory$
12: $p_{s,out} := \frac{w'_s \cdot p_s}{w'_m + w'_s}$
13: send $p_{s,out}$ to principal camera
14: **end for**

The Principal pixel block is responsible for calculation of the final HDR pixel value. Using the calibration data, the block reads the appropriate pixel from memory, multiplies it with the weight, and requests the weighted pixel from the secondary camera. The secondary camera is not necessarily connected to the same FPGA. Thanks to the Communication controller, where camera connection graph is stored, the secondary pixel is obtained [31]. The secondary pixel has already been multiplied by the HDR blending weight in the Secondary pixel block, thus only final addition is required. The resulting HDR pixel is further provided to the central unit.

The Secondary pixel block operates in the similar fashion. The block waits for the pixel request from the principal camera, reads the pixel from memory, multiplies by the weight and sends the value back to the principal camera. Both principal and secondary pixel blocks operate concurrently; hence, there is no wait time between principal and secondary pixel processing, which allows very fast calculation time and no loss in the frame rate.

Central Processing

The central FPGA acts as a global system controller. The received data comprises sixteen parts of the full HDR panorama, *i.e.* one part per SCIP. Besides pixel data, SCIPs send information about the correct position in the full-view panorama. Hence, the central unit decodes the position and places the HDR pixel at the appropriate location in the temporary storage memory. When all the pixels belonging to the same frame are received, tone mapping process starts.

The RGB pixel values are read from memory and transformed into the YUV color system, with 16-bit precision per channel. To be in accordance with the previous notation, the values of the pixel luminance channel Y will be denoted by L_w.

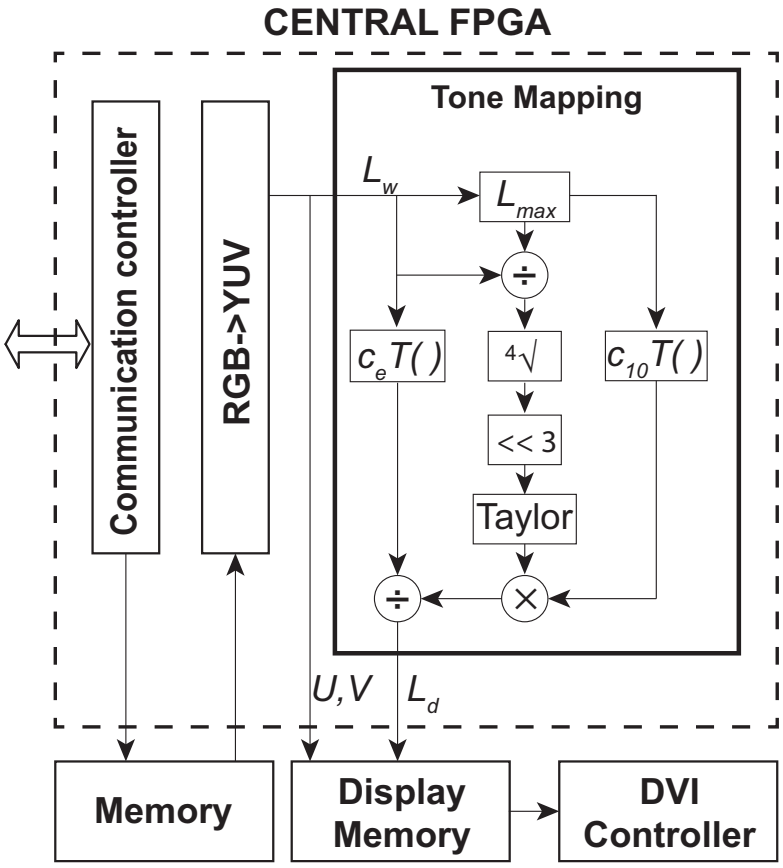
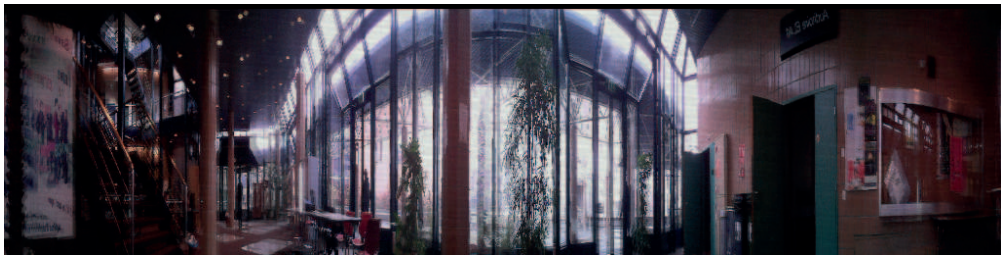


Figure 6.7: Internal architecture of the central FPGA. Tone mapping block is emphasized as the core processing unit.

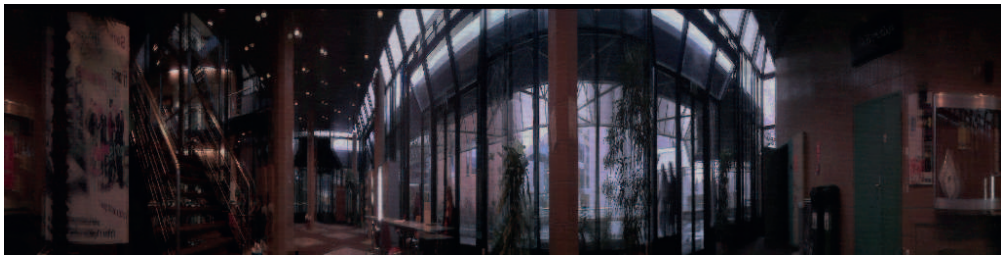
The tone mapping implementation consists of two parts: finding the maximum pixel luminance L_{max} and tone mapping curve implementation. Finding L_{max} consists of finding the maximum value in a sequence of the read luminances. L_{max} value is needed for the core tone mapping operation, as shown in (6.6). When HDR video stream is processed, L_{max} is taken from the previous frame, under the assumption that the scene illumination does not vary faster than response time of the HVS. The parameter is updated at the end of each frame.

Figure 6.7 presents the block diagram of the central unit, with emphasized tone mapping block. Chebyshev and Taylor polynomials are evaluated using pipelined implementation of the Horner scheme. The fast Anderson algorithm [68] is used for division implementation.

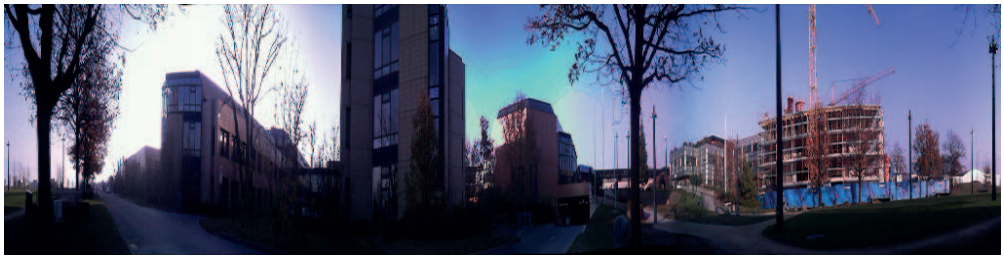
Taylor series approximation of the logarithm is fast converging only around the center point of the expansion, *i.e.* $x = 1$ when expansion from (6.7) is used. Even though the luminance value is in the range [0,1], the logarithm argument in the denominator of the tone mapping function (6.6) varies in the range [2,10]. Hence, the argument needs to be brought as close as possible to 1. Since the luminance values are logical vectors (vectors of ones and zeros),



(a)



(b)



(c)

Figure 6.8: Panoramic HDR reconstruction with a pixel resolution of 256 × 1024. The cameras were set (a) to automatic exposure mode, (b) such that two neighboring cameras have different exposure times, one four times shorter, and (c) one exposure time eight times shorter, to adapt to bright conditions of outdoor scenery. The blending weights are calculated using $\sigma_d = 300$, to provide sufficient influence of the secondary camera.

the identity (6.8) is used. The number of leading "ones" in the fixed-point representation of the luminance determines the scaling factor y, and the division is implemented as the arithmetical bit-shift-right operation.

$$\begin{aligned} \ln x &= \ln(x/2^y \cdot 2^y) \\ &\approx \ln(x/2^y) + y \cdot 0.6931 \end{aligned} \tag{6.8}$$

The tone mapped luminance value is combined with the corresponding chrominance components and written into the DVI controller for display.

6.2.6 Results and Discussion

Image Quality

The installed cameras have a vertical FOV = 46° and capture 4.9 Mpixels/frame in total. Even though the ratio of vertical and horizontal FOV of the system is 1:8, we experienced that a panoramic strip of size 256 × 1024 pixels provides enough pixel information, without significant deformation of the objects. This panorama is fitted in the VESA standard XGA frame (768 × 1024 pixels) and displayed directly on screen using DVI connection. The XGA frame is chosen due to 36 Mb capacity of the dedicated display memory in Figure 6.7.

In order to quantify the loss in image quality due to applied approximations, the peak signal-to-noise-ratio (PSNR) is calculated for images in the calibration set, whose subset is shown in Figure 6.3. The HDR image is created and tone mapped in Matlab using approximated and non-approximated calculations. Non-approximated double-precision tone mapped image is taken as the ground truth. Resulting luminance of the approximated tone mapping from Section 6.2.4 is quantized as a 16-bit value and its PSNR is measured to be 103.61 dB. Thus, luminance of the resulting image does not lose its original 16-bit precision.

Three video screenshots are shown in Figure 6.8. Figure 6.8a depicts an indoor scene using the automatic exposure mode of the cameras. The measured dynamic range is 1:43. Inside objects are well visible, however, the window region is saturated due to strong light outside of the room. Figure 6.8b shows the same scene rendered using the proposed HDR module. Even though overlap of FOVs is uneven for each camera pair, difference in color tone is not noticeable. Furthermore, the produced image shows details in previously saturated regions, such as the other buildings, while preserving visibility in the darker inside regions. The dynamic range of the reconstructed scene is increased to approximately 1:160, which results in 3.72 increase in dynamic range.

The indoor reconstructions suffer from ghosting of near objects due to parallax. The ghosting was expected, because the cameras were calibrated in an environment with no close objects. However, the observed ghosting is different from motion blur, which originates from the difference in exposures. Figure 6.8c shows a rendered HDR outdoor scene, where the closest objects were approximately at 30 m distance. Hence, the edges in this images are significantly sharper than in the indoor environment. The motion blur is not visible around the moving crane or tree branches, thanks to negligible difference in exposure times.

This HDR construction method does not provide as significant increase in dynamic range as some of the other methods, due to the use of only 2 f-stops. However, up to our knowledge, it is the only system which uses multiple cameras to create and render HDR radiance map simultaneously, and provides real-time HDR video signal at the output.

System Performance

The chosen figure of merit for performance of real-time systems is the total processing bandwidth, which best describes the system's capability. The figure of merit is calculated as:

$$BW = N_{pixels} \cdot F \cdot BPP \tag{6.9}$$

where N_{pixels} is the total number of processed pixels, F is the operational frame rate, and BPP is the number of bytes per processed pixel. As equations (6.1)-(6.3) show that all pixels acquired by the presented system are processed, the number of processed pixels is equal to sixteen VGA (640×480) frames. The operational frame rate is $F = 25\ fps$ as input and output frame rates are equal. Each pixel is represented with $BPP = 2$ bytes in RGB format. The conversion to YUV in the central FPGA transforms each pixel into two bytes for luminance, and one byte per chrominance channel.

Comparison of the designed prototype and algorithm implementation with the related systems is given in Table 6.1 and Table 6.2. The numbers in the comparison are taken from the original publications if they are published, or calculated by equation (6.9) using the available publication data.

Performance comparison shows that the proposed system is superior to the state-of-the-art systems for HDR video construction. The only comparable work is of Slomp and Oliveira [109] with 214 MB/s. However, this system uses the high-end GPU to implement only the tone mapping function. The main reason for high performance of our system is the fully pipelined operation which processes one pixel per clock cycle. Thus, frame rate is linearly dependent on the clock frequency.

Table 6.1: Performance comparison with the related full HDR systems.

Type	Full HDR systems			
	This work	[106]	[100]	[98]
Bandwidth [MB/s]	245.7	196.6	112	45
Processing unit	Virtex-5	Virtex-5	GeForce 680	–
Real-time video	Yes	Yes	Yes	Yes

Table 6.2: Performance comparison with the tone mapping systems.

Type	Full HDR systems				
	This work	[91]	[108]	[104]	[109]
Bandwidth [MB/s]	245.7	37.8	74	104.85	214
Processing unit	Virtex-5	Fire GL X1	GeForce 8800	Stratix II	GeForce 8800 GTX
Real-time video	Yes	No	No	No	Yes

Table 6.3: Slave FPGA device utilization.

Module	Total Used	Available
Slices LUTs	63732	69120
Slice Registers	40509	69120
BlockRAM/FIFO	89	128
DSP48Es	61	64

Table 6.4: Central FPGA device utilization.

Module	Total Used	Available
Slices LUTs	18376	69120
Slice Registers	17498	69120
BlockRAM/FIFO	88	128
DSP48Es	58	64

The utilization summaries of slave and central FPGAs are given in Tables 6.3 and 6.4. The utilization reports are provided for the complete system capable of supporting all forty-nine cameras.

Real-Time Video Examples

Two video examples of the real-time HDR reconstruction are recorded and provided as the supplementary material to this thesis. One example shows the improvement of our system compared to the automatic exposure mode of the used cameras. The second example shows the difference compared to manually reduced exposure time, in order to detect objects outside of the room. The blur around the object edges is due to parallax effect, which appears because the camera is calibrated for far objects. This issue can be resolved by having several different calibration parameter sets, which is one of the next steps in the platform development.

6.2.7 Conclusion

In this Section, we showed how a multi-camera system can be used for real-time HDR video recording. The system produces a real-time HDR video using multiple low-cost cell-phone cameras, *i.e.* without rather expensive HDR sensors. It is able to simultaneously acquire LDR data, reconstruct an HDR radiance panoramic composite frame, and tone map for realistic display on screen. High system bandwidth and 25 fps frame rate make this prototype an excellent choice for real-time and HDR video applications.

The reconstruction algorithm utilizes the overlap in FOVs of the camera sensors, which are set to different exposure times. We exploit this setup to increase the dynamic range of the captured images and construct an HDR composite image. The HDR image is tone mapped using the fast pipelined global tone mapping algorithm, which was adapted for efficient FPGA implementation.

6.3 Real-Time Object Tracking

6.3.1 Introduction

Object tracking has many applications in traffic monitoring [114], video surveillance systems [115], crowd analysis or even studying customers in malls. The data extracted by tracking can be used to observe the direction and the path taken by all objects moving through a scene. Tracking information is given by analyzing the differences between two or more consecutive frames. Using them, algorithms can not only analyze the speed variation and the size of a specific object in a scene, but also the areas where the object has moved. Tracking multiple objects in a high-resolution panoramic frame requires processing the large amount of pixels. The real-time applications imposes timing constraints on the used algorithm, which must be fast enough to process the whole frame. All items in the scene should be detected with the least amount of false recognitions.

GPUs are composed of hundreds of cores handling a large number of simultaneous threads. On the contrary, CPUs have less cores handling less threads at a time. GPUs are usually dedicated to the graphical tasks like pixel shading, rendering and accelerating graphics. The evolution of those devices has opened the possibilities of their use in other applications [116]. General-Purpose Computing on Graphics Processing Units (GPGPU) defines GPU used to perform computations that were usually handled by the CPU to achieve a faster execution speed [117] by exploiting parallelism. CUDA is the acronym for Compute Unified Device Architecture. This programming model and computing platform was developed by NVIDIA in 2006 to implement GPGPU. Currently, it is considered as the most mature application programming interface for those applications [117].

In computer vision, several state-of-the-art algorithms are used to detect moving objects [118]. One of them is the background subtraction method, which constructs a background model representing the scene without any objects. Each incoming frame is compared to this model to find the differences, which are considered to be the moving objects. In this section, we present a real-time implementation of a background subtraction technique and a connected component detector using a GPU. Furthermore, we introduce real-time processing techniques to cope with the detection of false positives.

The results of our object detection implementation are shown on high-resolution regions of interest from the omnidirectional videos. The final goal is to design an automated system for object detection in omnidirectional images that will scan through the frame and report “suspicious” behavior to the operator, without any human involvement. The idea is depicted in Figure 6.9.

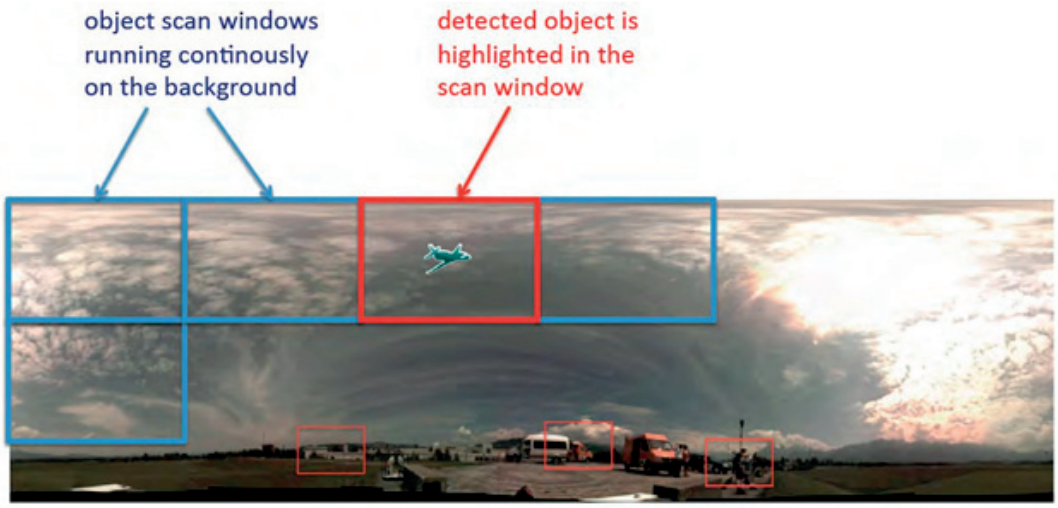


Figure 6.9: Illustration of the automated object detection scan system. The system detects a moving object in the panoramic frame, and displays the object's neighborhood on another screen.

6.3.2 Background Subtraction

There numerous background subtraction methods [119]. The first step of any background subtraction method is generating the background model, usually by taking the first few frames as a reference. The foreground pixels are extracted from each new processed frame, and compared to the corresponding pixels in the model. Using a connected-component detector, neighboring pixels are gathered in order to form the detected foreground objects. The new objects are compared to the ones formed in the previous frame to verify if the corresponding object has moved. We will present several background subtraction methods, starting from the simple ones to the more complex.

Simple Differentiation

Simple differentiation consists of computing the Metric Distance M_d between the color values of the current frame pixel and the color values of the background pixels at the same position (x, y). The value M_d is compared to a threshold T_{bgs}. If $M_d \leq T_{bgs}$, the pixel is considered as foreground since the RGB values of the frame pixel are different from the background pixel values. For a frame at time t, the metric distance is computed as:

$$M_d = (r - b_r)^2 + (g - b_g)^2 + (b - b_b)^2 \tag{6.10}$$

where r, g, b are the value of red, green, and blue color channels of the frame and b_r, b_g, b_b of the background.

The background model is updated to include the changes of the scene. The model stays unchanged for foreground pixels (x, y) to avoid pollution by pixels that do not belong to the background:

$$\mathbf{B}'(x,y) = \begin{cases} \alpha\mathbf{B}(x,y) + (1-\alpha)\mathbf{F}_t(x,y) & \text{For background } (x,y) \text{ pixels} \\ \mathbf{B}(x,y) & \text{Otherwise} \end{cases} \tag{6.11}$$

where α is the learning rate, **B** is the background pixel, and **F** is the foreground pixel..

Buffered Frames Actualization

Buffered frames actualization updates the background model using n_{frames} number of frames stored into a circular buffer memory. Each time a new frame is processed, it is added to this structure. A typical value of n_{frames} is between 4 and 10 depending of the video speed [120]. For each pixel, the value of $B(x, y)$ contains the mean value of the red, green, and blue colors of the frames contained in the buffer. It builds a background model resilient to abrupt changes of the scene. The metric distance M_d is computed as in the simple differentiation method and compared to the threshold T_{bgs}.

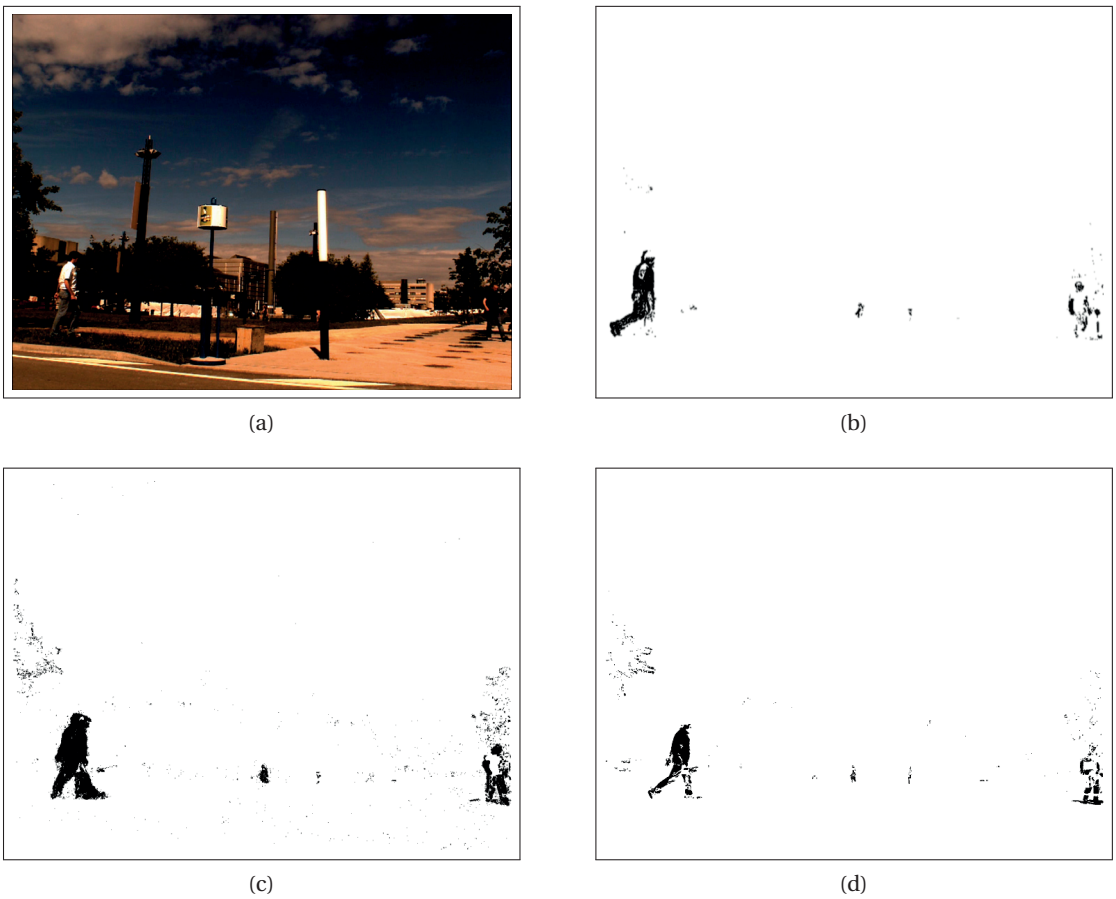
Single Gaussian

In this method, each background pixel is represented as a probability density function. The first n_{frame} frames are used to build this function. This method takes into account the noise present in the frame by modeling every pixel of the background with a single Gaussian distribution : $\backslash(\mu_t(x,y), \sum_t(x,y))$ where $\mu_t(x,y)$ is the average background color and $\sum_t$ is the covariance matrix for the pixel at position (x, y) at time t.

For each pixel, M_d is the log likelihood computed as [120]:

$$M_d = \frac{1}{2} log(2\pi)^3 \mid \sum_t(x,y) \mid + \frac{1}{2}(F_t(x,y) - \mu_t(x,y))(\sum_t(x,y))^{-1}(F_t(x,y) - \mu_t(x,y))^T \tag{6.12}$$

where $\mu_t(x, y)$ is a vector containing the red, green, and blue values of the average background color.



(a) (b) (c) (d)

Figure 6.10: (a) A captured scene from the original video sequence, and background subtraction results using (b) the simple differentiation method with $\alpha = 0.96$, (c) buffered frames with $n_{frames} = 5$, and (d) Gaussian background modeling.

The distribution is updated after each processed frame:

$$\begin{aligned} \mu_{t+1}(x,y) &= \alpha\mu_t(x,y) + (1-\alpha)F_t(x,y) \\ \sum\nolimits_{t+1}(x,y) &= \alpha\sum\nolimits_t(x,y) + (1-\alpha)(F_t(x,y) - \mu_t(x,y))(F_t(x,y) - \mu_t(x,y))^T \end{aligned} \tag{6.13}$$

where α is the same learning rate as in the simple differentiation method.

The results of applying the discussed methods are given in Figure 6.10.

6.3.3 Object Tracking

The aim of object tracking is to detect the movements of all objects in the scene by analyzing the variation of pixels from one frame to the next. It differs from object recognition where the algorithm looks for a predefined shape, *e.g.* a human face, over a scene [121]. Tracking an object can be done by following its specific point associations (Point Tracking [118]), its shape/contour (Kernel Tracking [118]) or its object region (Silhouette Tracking [118]).

The principal step of the algorithm is to detect objects that are not part of the background scene. Foreground objects consist of pixels that differ from the background's pixels. A pixel is considered as foreground if its color values have enough variations from the corresponding ones of the background model. However, the algorithm must be robust to the following situations.

Camouflage

A foreground object can have a similar color to the background scene. Distinguishing the object from the background in this case is harder since the pixel values composing the object are not different enough from the background ones. This implies that some of them may be categorized as background pixels. Detecting those pixels is done by reducing the detection threshold. However, reducing the threshold value leads to the detection of more background pixels as foreground. An appropriate value of the threshold must be determined in order to find a good trade-off depending on the scene.

Moving Camera

If a mobile camera is used to produce the video, or the camera is vibrating/shaking, the background image is not static, and updating is more difficult. This can lead to false positive detections. A possible solution would be to consider a block of pixels instead of processing individual pixels, in order to make the algorithm more robust. The average color of the block is more resilient to camera vibration.

Dynamic Environment

The background scene may have some objects that move and they should not be detected as foreground objects, *e.g.* waves, clouds, trees, rain. Furthermore, the movements can be irregular, *e.g.* traffic lights. Unless they are important for the tracking algorithm, these object should not be detected and tracked.

Shadows

The foreground objects can create shadows in the scene. The shadow cast makes the tracking of moving objects slower since more foreground pixels need to be analyzed. Also, the differentiation of the objects becomes harder since shadows increase the object region. Finally, if two shadows superpose each other, the two foreground objects will be considered as the same, since they share lots of connected pixels. Shadow removal algorithms are extensively studied in literature and detailed explanations can be found in [122].

Illumination Changes

The object detector must consider that the environment may change over time either progressively or suddenly. The learning rate should be chosen carefully to control the update of the background model, and to help resolving the abrupt variations in the illumination, *e.g.* fast moving clouds, or lights turned on.

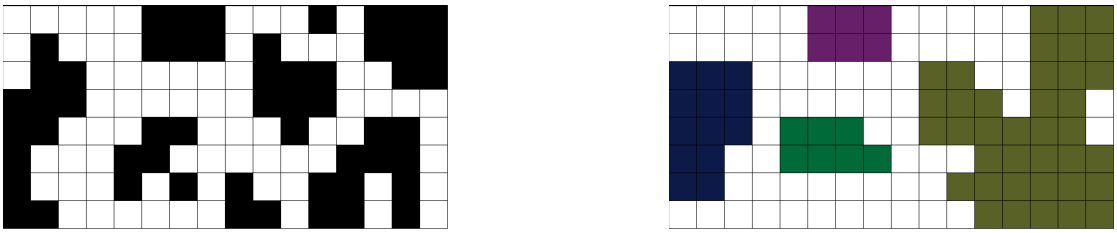
Noise

All image sensors produce noisy images. To remove the present noise from the frame, an image denoising algorithm should be used.

The object tracking algorithm based background subtraction object detection follows a series of processing steps. The first step is the aforementioned background modeling. The background model is updated with each new frame to adapt to the changes in the scene. Each presented method has its own way to compute and update the new background model, as explained previously.

The video frames usually contain unnecessary information that make the detection of foreground objects more difficult and prone to errors. The frame needs to be pre-processed in order to remove spurious information. Two main pre-processing steps are image denoising and shadow removal. In our implementation, a Gaussian filter is used for image denoising. Algorithms for removing shadows are implemented with a negligible increase in the processing time [122]. The shadow removal is done by removing all the pixels detected as foreground based on a small luminosity difference from the background model.

The resulting foreground mask often contains stains, *i.e.* undetected pixels inside the shapes of the foreground objects, or the shapes might not have the correct aspect. Morphological operators are used to correct and affine the shape. Dilation gathers the pixels by adding foreground pixels to frames. Only background pixels are analyzed in this case. A window of size s_{window} is created around each pixel. If more than the specified threshold $T_{dilation}$ of pixels inside the window are foreground, then the pixel is considered as the foreground. In opposite, erosion removes the isolated pixels by erasing the foreground pixels. Similar to dilation, a window of size s_{window} is used. If more than the specified $T_{erosion}$ of neighboring



(a) (b)

Figure 6.11: (a) An illustration of the output of the background subtraction algorithm, and (b) the labeled object after morphological operations.

pixels are background, then the pixel is considered as background. For both algorithms, the larger the size s_{window}, the more effective dilation/erosion is. High value of the threshold reduces the dilation/erosion influence. Dilation followed by erosion will fill the holes in the shape of the foreground mask. Erosion must be done before dilation in order to remove the small shapes in the foreground mask.

An illustration of the effects of the morphological operations is shown in Figure 6.11. The unconnected components are processed to form compact objects that represent the foreground pixels. Following their extraction, the objects are labeled and uniquely identified. At this point, the objects are ready to be compared to the objects being tracked, *i.e.* the ones detected in the previous frames. If two objects have a similar size, color and their positions are close to each other, they are considered as the same object, and its position is updated. Otherwise, a new object is created in order to be tracked in the following frames.

6.3.4 Experimental Results

The single Gaussian algorithm is implemented, because of the best obtained results in our experiments. The program is run on a GeForce740M GPU with 2048 MB of memory, and 980 MHz clock frequency.

The frames in Figure 6.12a are the resulting foreground masks after the object extraction algorithms, and Figure 6.12b shows the scene with the detected objects being tracked. All the moving objects are detected. However, a false positive represented by the white rectangle can be observed. The false positive corresponds to the top of a tree moving in the wind.

6.3.5 Conclusion

The presented background subtraction techniques are suitable for GPU implementation. The algorithm used to extract the foreground pixels or to build the background model can be parallelized, where the GPU processes each pixel using separate threads. Almost all steps of object tracking can be run in parallel to achieve better results compared to the sequential

(a) (b)

Figure 6.12: Results of the object detection and tracking represented by (a) foreground masks, and (b) rectangles around objects in the original frame.

implementation. A crucial part of using a GPU for object tracking is to pre-process the scene with image processing algorithms. It improves the results by removing the false positives and standalone pixels that significantly increase the execution time. Furthermore, choosing the appropriate algorithm with the correct parameters for background subtraction is crucial.

The presented implementation is a GPU implementation, and it can be run on any PC. *GigaEye II* camera system can be connected to the PC via the frame grabber that converts the HDMI output of *GigaEye II* to USB3. The object tracking code includes the frame grabber stream as one of the possible inputs and it can automatically track objects in the received omnidirectional image.

7 Conclusion

In this thesis, we presented a new approach to designing multi-camera systems, which consists of merging the image sensors and the processing units into a single system. Unlike state-of-the-art systems that use cameras purely for image acquisition, and a setup of PCs for its offline processing, the two presented camera systems use FPGAs for real-time image processing. We presented the complete design flow including the hardware design choices, algorithm development for panorama construction, FPGA implementation, user interface options, and finally, real-time tests.

The first contribution of this thesis is development of a hardware-suitable real-time panorama construction algorithm. Most of the currently used stitching and blending algorithms are developed and optimized for running on a CPU or a GPU. Their direct hardware implementation is either not fast enough for real-time operation, or does not provide acceptable image quality. We presented a novel image blending algorithm called Gaussian blending, which builds up on a well-known alpha blending. The implementation of Gaussian blending on our camera system provides seamless panoramic image, and suppresses appearance of ghosts. Furthermore, Gaussian blending can be implemented in real-time for any number of cameras in the system without loss of image quality. The disadvantages and limitations of the Gaussian blending are that the variance of the used Gaussian curve has to be determined manually, and that the ghost suppression around objects sometimes results in slightly blurry edges.

The second contribution is the full design of a miniaturized multi-camera system called *Panoptic* . The system is built using fifteen commercially available low-cost cell-phone cameras placed on a hemispherical dome. The real-time image processing is performed on a custom-made Virtex-5 FPGA board. The panorama construction is implemented using the Gaussian blending and provides good quality results. Furthermore, we demonstrated the ability of the camera to create the environment for virtual reality goggles Oculus Rift, which gives an immersive experience to the user. Up to our knowledge, this is the first camera-goggles system that can also be used where telepresence is required.

The third, and the most important contribution is the design of a high-resolution multi-camera

system, *GigaEye II* . This system also works in real-time, thanks to the novel, distributed panorama construction. The system is designed as a multi-layered processing system, with each processing unit dedicated to reconstructing the partial FOV. Using such approach, it is possible to process and support very large data rates. The system can currently process all 320 Mpixels that are captured each frame. The system is designed to be modular, and the increase in resolution can be achieved by placement of the new cameras and addition of one FPGA processing board per four cameras.

Finally, the last chapter of this thesis presents possible of the designed multi-camera systems.

7.1 Future Prospects

The future of the real-time computational imaging systems can be discussed in the following groups:

- **Algorithms:** The proposed Gaussian blending for *Panoptic* and multi-band blending for *GigaEye II* provide high quality results, but are not optimal solutions. Recently, several new methods were developed that make use of the optimization algorithms. Implementing an optimization in real-time on FPGA is a very challenging and interesting problem, which has not been solved up to now. Solving it would definitely make a significant impact in real-time computational imaging, as well as many other research fields.
- **Miniaturization:** *Panoptic* camera utilized compact camera modules to reduce the system size. However, the processing PCB and FPGA are large with respect to the dome where the cameras are placed. The design of an ASIC to replace the FPGA, reduce the number of external components, and reduce the PCB size is the next logical step. Furthermore, the design of an ASIC with certain level of programmability can lead to commercialization of *Panoptic* as a consumer product.
- **True Gigapixel video:** The finalization of the true Gigapixel real-time omnidirectional sensor. The “upgrade” to higher, Gigapixel resolutions has already been envisioned, and its realization is only a matter of need for such a high-resolution system.
- **New applications:** The new applications can be proposed for any of these real-time multi-camera systems, such as 3D cinematography, video super-resolution, embedding virtual reality content in the scene (for gaming).
- **Displays:** A major drawback for high-resolution camera systems is the inability to display the full image, and demonstrate its full potential in real-time. Hopefully, the fast-growing advancements in the camera technology will be followed by very high-resolution displays in the near future.

A GPU Implementations of the Panorama Stitching

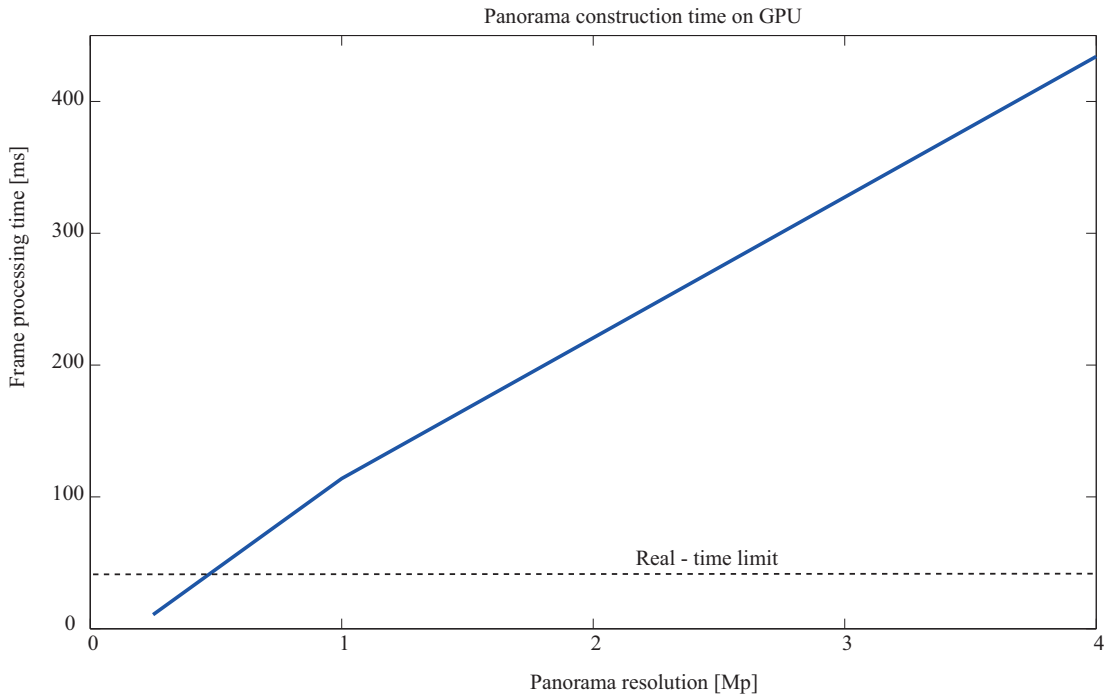


Figure A.1: Panorama construction time with respect to its resolution. The program was run on the GeForce740M GPU.

B PixelPlus PO4010N Sensor Module Specifications

Table B.1: Full PO4010N characteristics.

Parameter	**Value**
Total Pixel Array	386 × 320
Effective Pixel Array	368 × 304
Pixel Size	3.6 μm× 3.6 μm
Effective Image Area	1.325 mm× 1.094 mm
Clock Frequency	9 MHz
Dark Signal	0.1 mV/sec
Sensitivity	1.47 $V/(lux \cdot s)$
Saturation Level	1200 mV
Dynamic Power Consumption	41.15 mW
Operating Temperature	-30 .. 70 $^{\circ}C$
Dynamic Range	99 dB
SNR	42.5 dB
Filter	RGB Bayer color filter
Data Interface	8-bit parallel + synchronization + clock
Frame Rate	25 fps
Maximum Output Resolution	352 × 288 (CIF)

C CMOSIS CMV20000 Sensor Specifications

Table C.1: Full CMOSIS CMV20000 image sensor characteristics.

Parameter	**Value**
Total Pixel Array	5124 × 3844
Active Pixel Array	5120 × 3840
Pixel Size	6.4 μm× 6.4 μm
Imager Size	32.77 mm× 24.58 mm
Clock Frequency	480 MHz LVDS
Temporal noise	8 e-
Pixel Type	Global shutter
Dark Current Signal	125 e/s
Dynamic Power Consumption	1100 mW
Operating Temperature	-20 .. 70 °C
Dynamic Range	66 dB
PRNU	1 %
Filter	RGB Bayer color filter
Output Format	12-bit
Data Interface	16 LVDS data channels + 1 LVDS control line + 1 LVDS DDR output clock
Frame Rate	up to 30 fps

D Scalability of *Panoptic* Camera

Each FPGA board can interface with 20 cameras. To support a higher number of cameras and increase the throughput of the Panoptic camera, multiple FPGA boards must be incorporated. Hence, the omnidirectional view reconstruction workload is distributed and the algorithm operates in parallel on all FPGA boards. Thus, a central FPGA is required to receive the output data from all FPGA boards, apply the final blending process and transfer the result to a PC for display.

A scalable FPGA-based system is devised, using the designed FPGA board, to support the application development of the Panoptic camera. The devised system consists of four layers: 1) image sensors, 2) FPGA boards handling local image processing, 3) one central FPGA board for control, external access and last stage image processing, 4) a PC in charge of the applicative layer consisting of displaying the operation results transmitted from the central FPGA board. The designed central board supports up to five layer-2 FPGAs. Figure D.1 depicts the devised architecture for a typical Panoptic system.

The layer-2 FPGA boards implement the architecture presented in Section 4.5. The outputs of these boards carry the value related to locally blended pixel values and their corresponding weight. These two 16-bit values are streamed to the central unit for the final blending step via an LVDS link. The LVDS link is implemented in the Data and control unit shown in Figure 4.3. The 16-bit pixel value and its weight are split into the most significant byte (MSByte) and the least significant byte (LSByte). Xilinx embedded serializer blocks are used to serialize the bytes and transfer them to the central FPGA. The byte order is as follows: 1) LSByte of the pixel value, 2) MSByte of the pixel value, 3) LSByte of the blending weight, 4) MSByte of the blending weight.

The full-resolution frame is transferred via LVDS, irrespective of the FOV of the cameras connected to the observed layer-2 FPGA. In practice, this means that the pixels in the reconstructed image which are not observed by the connected cameras are also transferred. In such cases, both the pixel value and the weight are set to zero, *i.e.* the pixel is considered purely black and as such the least influential in the final blending operation.

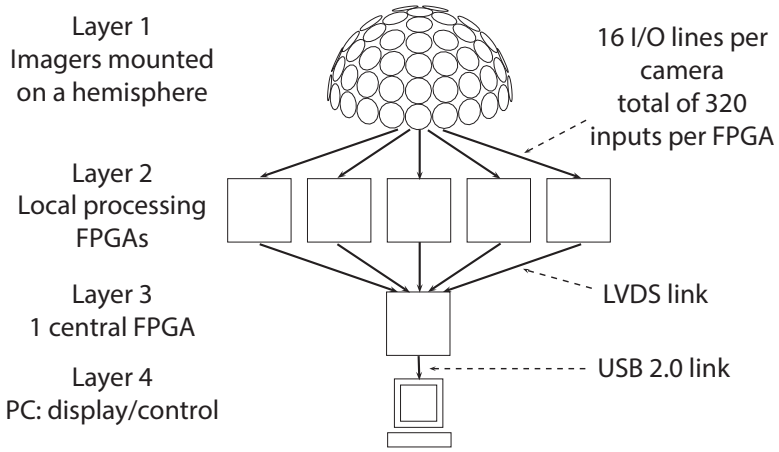


Figure D.1: Architecture of the multi-layer Panoptic system.

Furthermore, the LVDS link is used to transfer commands issued from the central to the slave FPGAs. The implemented commands are “start/stop video stream”, “capture a single snapshot” and “reset the whole system”.

The central FPGA architecture consists of two main parts: input buffers that store data from the slave FPGAs and the image processing unit. The input buffers deserialize the incoming data and recover pixel values with its respective weights. All slave boards are synchronized a with maximum of one clock cycle latency. Hence, short input FIFOs are used as input buffers and memory storage is avoided. The processing unit of the central FPGA is significantly simpler than in the slave FPGA, as it only contains the Blending module. It calculates the final results based on the pixel values and the weights calculated in the slave FPGAs. The final values are sent to the PC for display, via a USB link.

E *Panoptic* Graphical User Interface

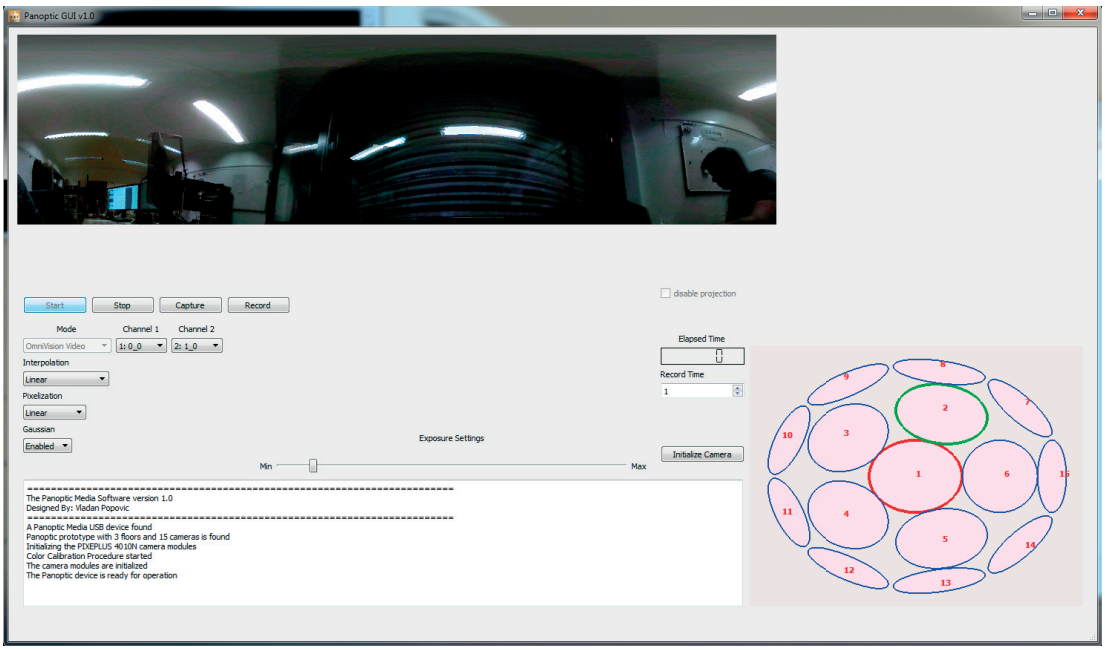


Figure E.1: Graphical User Interface for controling and visualizing data from *Panoptic* camera.

The Graphical User Interface (GUI) for the *Panoptic* camera is shown in Figure E.1. The screenshot shows the setup options for running the camera. Except start, stop, capture, and record commands, the user can choose the blending method, discretization scheme, and the exposure time setting. The user can also run the camera in a different mode, such as streaming a single camera video, when the camera can be chose by clicking on the desired circle in the bottom right corner. The user sees the real-time panoramic reconstruction at the top, as well as on the virtual reality goggles Oculus Rift if they are attached to the PC.

F Thermal Camera Test of the CMV20000 Headboard PCB

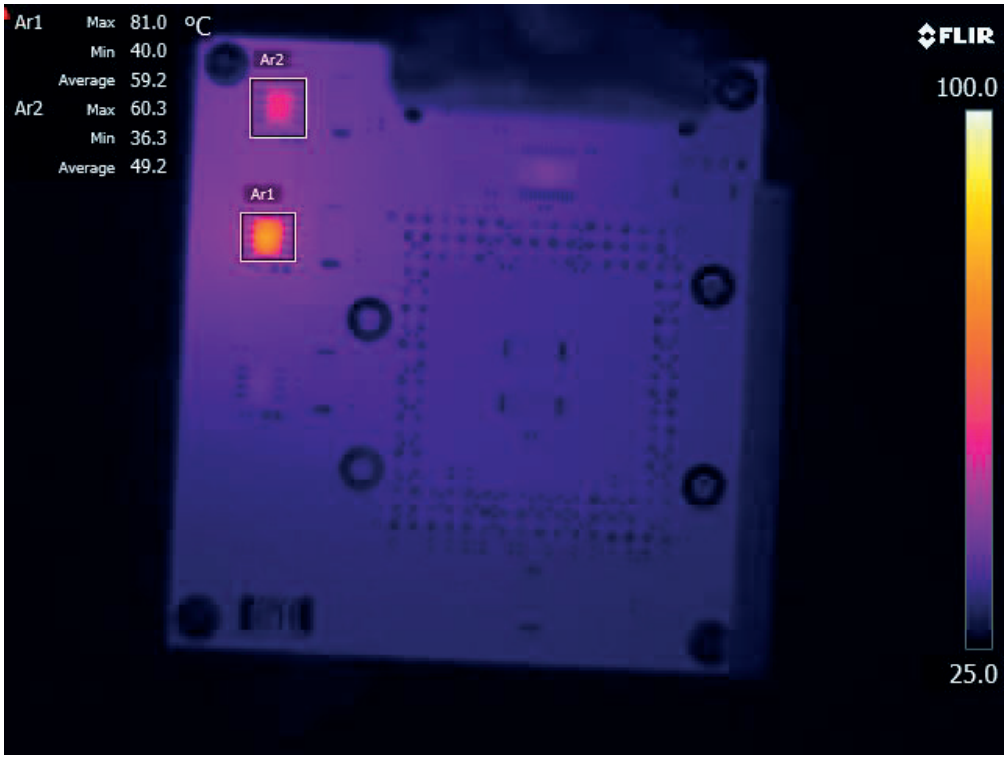


Figure F.1: Thermal camera shots of the back side of the CMV20000 headboard PCB. After a few minutes of operation, the DC/DC converter was reaching its operational limit of 80 °C.

G Layouts of the Designed PCBs

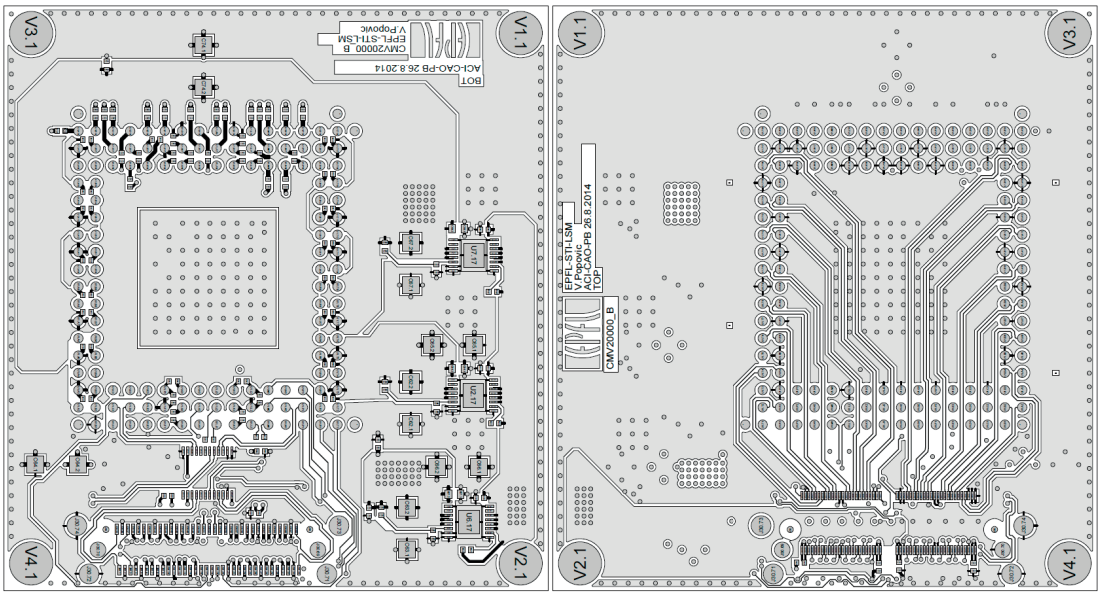


Figure G.1: (left) Bottom and (right) top layout of the designed headboard PCB for *GigaEye II* .

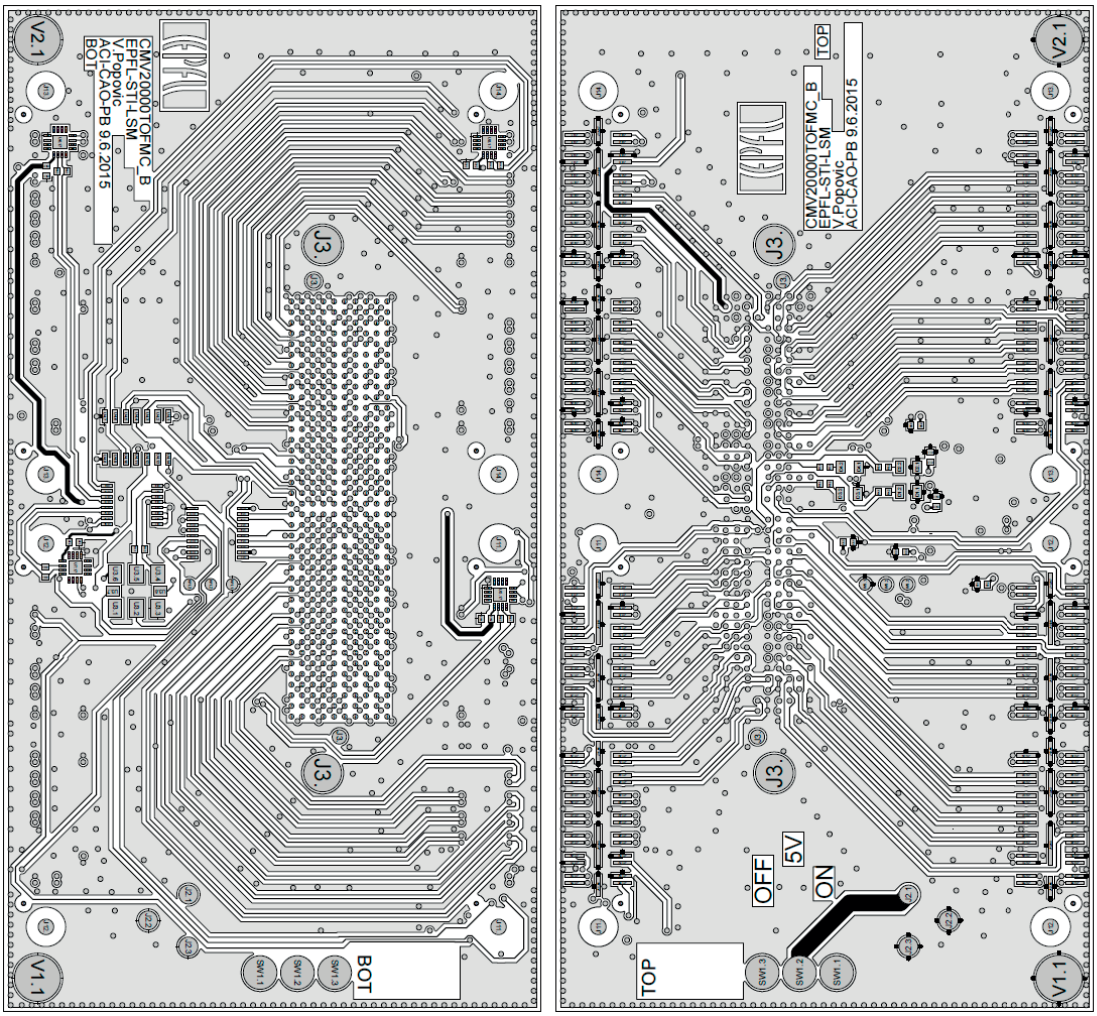


Figure G.2: (left) Bottom and (right) top layout of the designed FMC interconnection PCB for cluster FPGA of *GigaEye II* .

H *GigaEye II* Technical Drawing

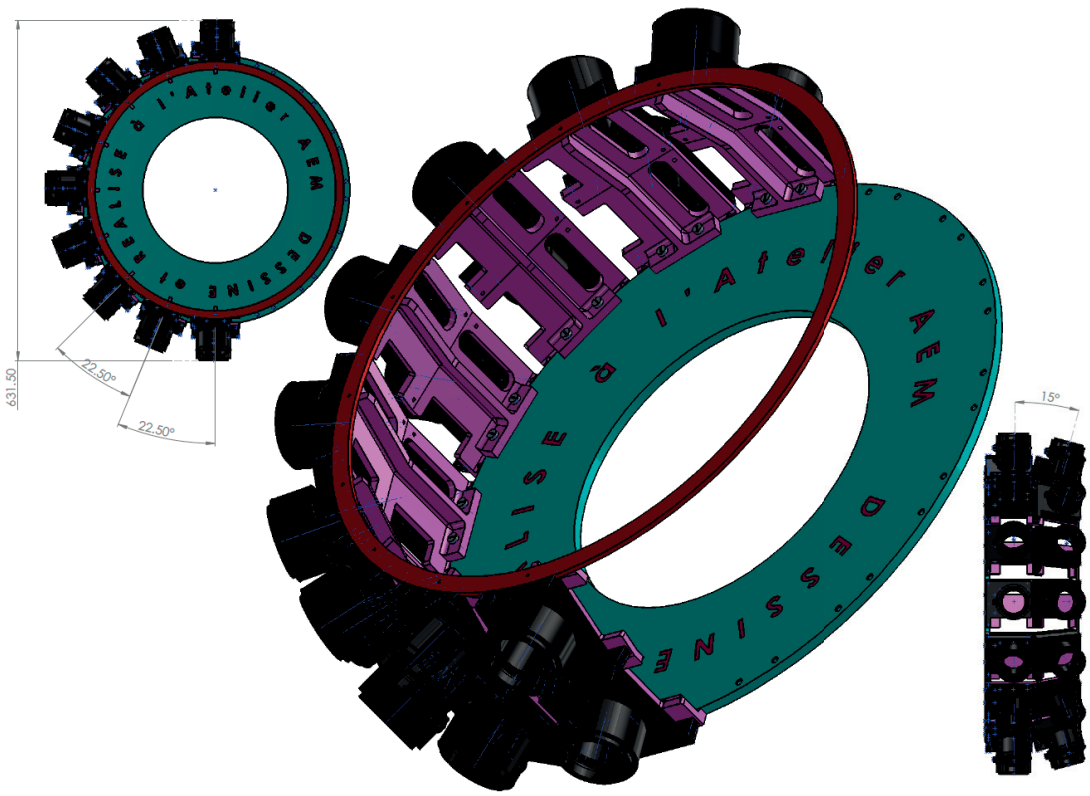


Figure H.1: Full size technical drawing of the *GigaEye II* system.

I Publication Print-outs

The manuscripts of significant publications that were not extensively utilized in the thesis are given in this appendix. They present additional and more thorough research results.

Original article published in Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vancouver, Canada, 2013

REAL-TIME HARDWARE IMPLEMENTATION OF MULTI-RESOLUTION IMAGE BLENDING

Vladan Popovic, Kerem Seyid, Alexandre Schmid, Yusuf Leblebici

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Microelectronic Systems Laboratory
Lausanne, Switzerland

ABSTRACT

A novel real-time implementation of a multi-resolution image blending algorithm is presented in this paper. A multi-resolution decomposition of the input is used to blend multiple images at different scales. Processing time is shortened by designing a pipeline system. The proposed solution requires less hardware multipliers and is able to achieve very high operating frequencies, compared to the current designs. The presented hardware architecture is optimized to support multiple simultaneous video streams, and high frame rates at High-Definition (HD) resolutions.

Index Terms— Real-time systems, Pipeline processing, Image fusion, Image decomposition, Field programmable gate arrays

1. INTRODUCTION

The limited angle of view of modern cameras has created the need to combine two or more images into a single one, in order to increase the effective angle of view. The creation of panoramas or image mosaics has been a popular research topic over the past years. The problems which must be solved relate to the proper image alignment and seamless image blending.

The purpose of the image alignment is to determine the correct orientation and position of the original images in the final mosaic. Various algorithms for aligning the captured images were developed [1], [2], [3]. Additionally, it is possible to reconstruct a panoramic mosaic using a video stream of frames [4]. While image alignment processes the geometry of the image, blending algorithms handle the pixel intensity in the final mosaic.

A major issue in creating photo-mosaics resides in the fact that the original images do not have identical brightness levels. This may be caused by diverging camera orientations in space. Thus, cameras acquire more light in some of the shots. The problem manifests itself by the appearance of a visible seam in regions where the images overlap. The blending algorithms based on a weighted average between pixels in every image, *e.g.* "Cut and paste" algorithm [4], can reduce or even completely remove the seams. However, the drawback of a weighted average lies in a high frequency blurring in the presence of any small image alignment error. A possible solution to this issue consists of using a multi-resolution blending algorithm [5], [6] where high frequencies are combined in a small spatial range, thus avoiding blurring.

Blending is usually realized as a post-processing operation on a Personal Computer (PC). However, real-time blending is often required in multi-camera systems, *e.g.* [7], [8]. Real-time operation can be a very challenging problem. Hence, a Graphics Processing Unit (GPU) implementation or a dedicated hardware solution are often considered. Various existing GPU implementations of multi-resolution blending algorithms [9], [10] and their performance will be compared to this work. On the other hand, Field Programmable Gate Arrays (FPGA) are widespread used platforms, that enable fast development. Sims and Irvine [11] designed an FPGA system for blending using gradient pyramids. However, their system targeted blending greyscale images with VGA resolution. Furthermore, the system had large memory requirements, because all of the temporary results in the calculation process had to be stored. Song et al. [12] introduced a resource-efficient three-stage pipeline processing system. Still, their system only supports dual channel image fusion and is also constrained to greyscale images with VGA resolution. Finally, Van Der Wal and Burt [13] designed an Application Specific Integrated Circuit (ASIC) able to decompose an input image into multiple resolutions. However, multiple processing and memory chips have to be used in order to blend images.

In this paper, a novel real-time FPGA-based implementation is presented. The dedicated hardware for the multi-resolution blending algorithm is implemented using a fully pipelined architecture. The requirements for storage elements is reduced since only the final results are stored into memory. Furthermore, the design is able to support higher frame rate and higher image resolution than earlier proposed systems.

The outline of the paper is as follows. An overview of the multi-resolution image blending is given in Section 2. The formulation of the problem and proposal of the new implementation are explained in Section 3. Finally, the experimental results and comparison to the related work are presented in Section 4.

2. MULTI-BAND BLENDING

Multi-Band Blending (MBB) [3] is based on a multi-resolution decomposition of the original images and their blending across octave frequency bands. The images are represented using a Laplacian Pyramid (LP) [5], as it has perfect and simple reconstruction [14]. Several steps are performed to obtain the desired LP. The image is first blurred and then downsampled by a factor of 2 to obtain a low-pass image. The low-pass filter proposed in [14]:

$$H(z) = G(z) = \frac{1}{16}(1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + z^{-4}) \quad (1)$$

has a very high precision, since it uses only integer coefficients. Furthermore, this filter can be implemented in hardware using only adders and shifters, which is further detailed in Section 3.

The low-pass image is then upsampled by 2 and reconstructed using an interpolation filter. The interpolation filter, in this case, is

The authors gratefully acknowledge the support of XILINX, Inc., through the XILINX University Program.

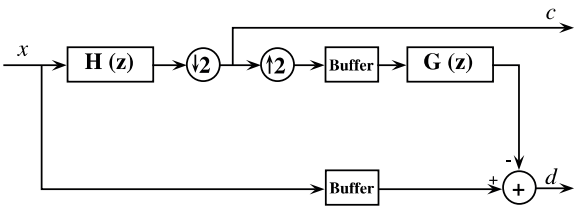


Fig. 1. Two-level LP decomposition

identical to (1). The interpolated image is subtracted from the original to determine a high-frequency version of the input image. The LP is created by repeating the same procedure on the downsampled low-frequency image.

The regions of overlap between images may be of an arbitrary shape. Thus, a mask should be created, defining the pixels which should be taken from the original image and their respective weight. A binary mask is assigned to each image, where 1 represents a pixel that should be taken from the selected image. This mask is further decomposed into a Gaussian Pyramid (GP), which is created by repetitive blur and downsample operations, *i.e.* each level of the pyramid is a low-pass version of the previous level. Brown [5] and Burt [3] suggest to use the same filter for the generation of the GP weight mask as for the LP. The use of the same filter simplifies the system and provides seamless blending results when the overall brightness level of the images does not differ significantly.

Each frequency band of the LP is combined with the respective frequency band of the other LPs, *i.e.* other images. A weighted average is applied within the overlapped areas, which are proportional in size to the wavelengths represented in the band. Hence, when coarse features occur in the overlapping region, they are gradually blended over a relatively large distance, without blurring or degrading finer image details in the neighborhood [5]. The weights are taken from the corresponding mask GP. In case of blending two images, A and B, the blending of one pyramid level is expressed as:

$$I(x,y) = I_A(x,y)w(x,y) + I_B(x,y)(1 - w(x,y)) \qquad (2)$$

where I_A and I_B are pixel intensities and w is the pixel weight.

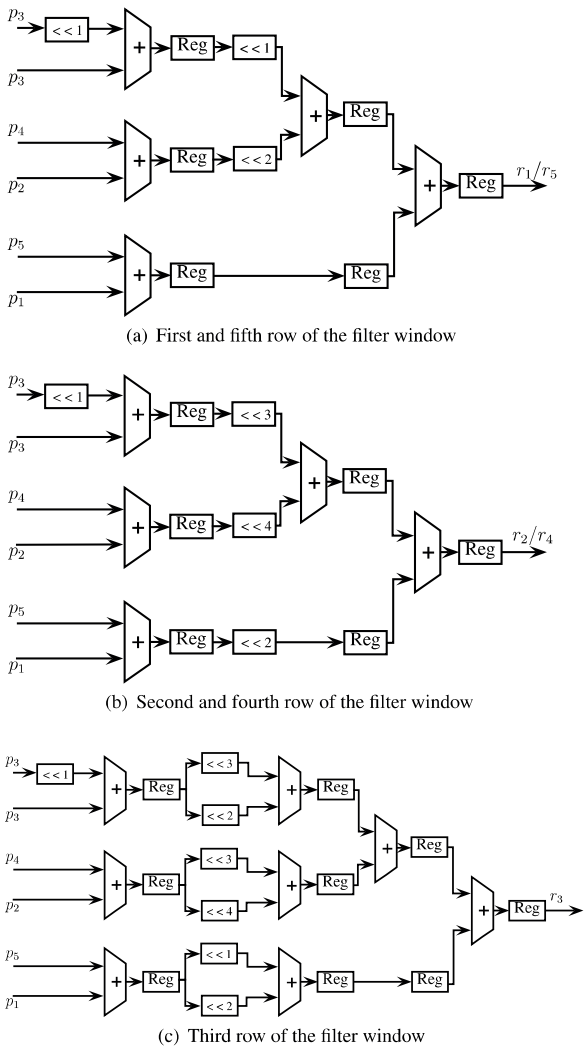
3. FPGA IMPLEMENTATION

3.1. Laplacian Pyramid Decomposition

In this paper, we propose an FPGA design of the multi-resolution image blending based on LP decomposition. A fully pipelined architecture is utilized. Figure 1 shows the data flow diagram of two-level LP decomposition. The obtained results are coarse (c) and detail (d) images. Filters from (1) can be expressed in the spatial domain by the matrix:

$$H = G = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad (3)$$

Direct two-dimensional filtering is implemented, since it requires fewer buffers for storage of temporary results than separable filtering. Even though a 5×5 pixel window is needed for filtering with $G(z)$, at least two rows (columns) are filled with zeros after the upsampling operation. Hence, the buffer following the upsampler in



(a) First and fifth row of the filter window

(b) Second and fourth row of the filter window

(c) Third row of the filter window

Fig. 2. Low-pass filter implementation

Figure 1 only stores two rows (columns). The interpolation filtering starts when the third non-zero row (column) pixels are arriving from the pipeline. The buffer in the lower branch in Figure 1 has the same depth as the buffer located in the upper branch, and acts as a delay element which synchronizes the original image with the interpolated pixels. The full decomposition into an LP is realized by cascading the proposed implementation. The number of the cascaded blocks corresponds to the desired number of LP levels of decomposition.

The acquired images are temporarily stored into a Random Access Memory (RAM). The LP decomposition and filter implementation depend on the order of the pixels which are read from the RAM. The first five pixels of each row are read consecutively, *i.e.* five pixels are read from the first row, followed by five pixels from the second row, etc. Subsequently, the filter window is shifted by two columns to the right. By reading from the memory in this manner, both low-pass filters $H(z)$ and $G(z)$ in Figure 1 provide outputs column-by-

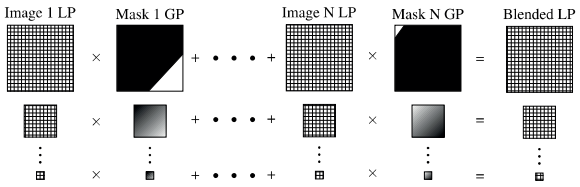


Fig. 3. Illustration of MBB

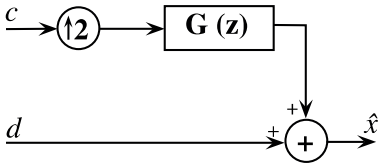


Fig. 4. Two-level LP reconstruction

column, pixelwise. The analogous implementation is also possible, where reading is done column-by-column, and the output pixels are provided row-by-row. However, most of the standard image resolutions have higher horizontal than vertical resolution. Hence, the first option is chosen, because the buffers in Figure 1 can be significantly smaller when storing two columns, instead of two rows. To cancel the edge effect at the image boundaries, the edge extension is performed by reflecting two rows (columns) across the edges.

Figure 2 shows the implementation of the filter block. It can be observed from (3) that pixels in the first and the fifth row of the window are multiplied with the same coefficients. Hence, the same hardware architecture can be used in both cases. The similar situation occurs with the second and the fourth row, with different coefficients. In Figure 2, signals p_1 - p_5 denote the intensities of pixels in the columns 1-5. Output signals r_1 - r_5 denote the values of the filtered rows. To obtain a final filtered value of the pixel, r_1 - r_5 are summed.

An important benefit of this implementation lies in the absence of any hardware multipliers. Multiplications by 2, 4 and 8 are realized by binary shifts to the left by 1, 2 and 3 bits, respectively. The only multiplicand which is not a power of 2 is 3 and it is obtained by adding the operand to its double, *i.e.* shift by 1.

The operating frequency of the system is increased by placing pipeline registers following each addition. The registers are, however, not needed after shift operations, since logical shifts do not require any processing time. The advantages of this pipeline architecture in terms of system performance is shown in Section 4.

3.2. Blending

In addition to LP decomposition, MBB requires a weight mask for each LP level of the image, as explained in Section 2. Weight mask GPs are pre-calculated and stored in a RAM, since their size is much smaller than the size of the original image. The first level of the pyramid is filled with only 1 and 0. Hence, if the image resolution is $K \times M$, the lowest level of the GP occupies $K \times M$ bits in the RAM. The weights in the second level of GP can be represented with 4 bits. Since the resolution of the second level is $\frac{K \times M}{4}$ pixels, the total bit size is the same as the first level. This rationale can be applied to all following levels. Hence, the total size that the GP occupies in RAM is $K \times M \times L$ bits, where L is number of levels in the pyramid.

Figure 3 illustrates the process of MBB. N images are simultaneously decomposed into their respective LPs in N parallel branches. The process of decomposition is synchronized in a manner to avoid storing the LPs in the RAM. Weights are read from the memory and multiplied by the corresponding coefficient in the LP. The weighted coefficients from each image, *i.e.* parallel branch, are summed to form a blended LP. The computation of the blended LP is also realized in a pipeline, with registers following each multiplier and adder.

Different LP levels cannot be simultaneously blended. The highest level, *i.e.* the one smallest in size, is blended later, since the LP level is obtained last. Hence, the blended LP levels are not obtained at the same time and they have to be stored in the RAM. These are the only intermediate results that have to be stored.

3.3. Laplacian Pyramid Reconstruction

Reconstructing the resulting LP is realized as shown in Figure 4. The LP coefficients are read from the RAM, starting from the highest level, *i.e.* low-pass image. The coefficients are read in the same manner as presented earlier, *i.e.* row-by-row. The same reconstruction filter $G(z)$ from (3) is used. The coarse image (c) is upsampled and interpolated to increase the image resolution. Afterwards, the detail image (d) is added. The resulting image ($\hat{x}$) is used as a coarse image input to the next level of reconstruction.

4. EXPERIMENTAL RESULTS

The hardware design is implemented on a Virtex-7 FPGA development board VC707, with 1 GB of external 800 MHz Dual Data Rate type 3 (DDR3) RAM. The maximum synthesizable operating clock frequency in the design is 420 MHz. The tightest constraint on the clock frequency is imposed by the adders in Figure 2. It is important to note that the proposed design is driven by only one clock signal. In [11] and [12], each pyramid level is driven by a different clock; each higher level in the pyramid is decomposed using a four times slower clock signal. Having only one clock domain is especially important if the design is to be fabricated as an ASIC, where clock routing becomes complicated in case of multiple clock trees.

The implemented design supports a dual video stream, but it can be extended to support more cameras, if it is needed. The data from the camera is recorder in RGB format, with 8 bits depth per color channel. An LP decomposition is done for each color channel, and they are operating in parallel. Each frame is decomposed into 4 LP levels, which is the maximum possible number for the chosen resolution. The display video resolution is 1920 × 1080 (HD 1080).

Table 1 shows the FPGA resource utilization summary and comparison to related work. In this work, the external RAM is only used

Table 1. FPGA resource usage comparison

	This work	**[11]**	**[12]**
Resource		**Used**	
Slices	7467	13287	2641
BlockRAM	14	430	38
DSP	4	–	–
External RAM [MB]	8.26	1.22	1.20
Family	Virtex-7	Virtex-2	Virtex-4

(a) (b)

Fig. 5. (a) Prealigned captured images and their masks; (b) photo-mosaic of EPFL campus created using the proposed MBB implementation

Table 2. Timing performance comparison

	This work	[11]	[12]	[13]
Max. freq. [MHz]	420	31	–	20
Frame rate [fps]	94	101	25	55
Resolution	HD 1080	VGA	VGA	512×512
Pyramid levels	4	4	3	10
Pixel depth [bits]	24	8	8	8

for storing mask GPs and the resulting LP. Larger external RAM occupancy is only due to the increased image resolution and color images. A reduction in BlockRAMs for temporary data storage and in used FPGA slices is observed. The design uses more slices than [12] because of the difference in filtering implementations, and one more LP decomposition level. In this work, filtering is realized using only adders made of Look-Up Tables (LUT) and registers, instead of multipliers in DSP blocks. The data from related work is taken from the original publications. Unknown information is represented by the "–" sign in all tables.

Table 2 shows the timing performance of blending two video streams. The maximum operating frequency is much higher than observed in the previous implementations. Apart from the newer FPGA family, the speed improvement is further influenced by a fully pipelined computation architecture. The achieved frame rate of 94 fps is very close to the best performance of the related systems [11]. However, the proposed design achieves this frame rate for significantly higher display resolution.

A comparison with GPU implementations is given in Table 3. The FPGA implementation is superior to the GPU. The GPU solutions are not able to achieve frame rates higher than 2 fps for display resolutions of more than 1 MP. Furthermore, the proposed architecture has a constant processing speed independent of the amount of overlap between the images. This is an important advantage compared to the possible software implementations.

Table 3. FPGA vs. GPU performance comparison

	This work	[9]	[10]
GPU	–	GeForce 8	Quadro 4600
Frame rate [fps]	94	0.43	1.79
Resolution	HD 1080	1147×608	1024×1024
Pyramid levels	4	–	7
Pixel depth [bits]	24	24	24

Figure 5 illustrates result of the proposed design. The images in Figure 5(a) are taken using two Aptina MT9P031 5MP sensors. The images are prealigned on a PC and stored into memory of VC707 Xilinx development board. The first level of the GP is also shown in Figure 5(a). The resulting blended image is shown in Figure 5(b).

5. CONCLUSION

In this paper, we propose a fast multiple-image blending hardware implementation. The blending algorithm is based on a multiresolution decomposition into an LP and image blending in different frequency bands. The proposed pipeline implementation is faster and less resource-demanding than the previous solutions. The experimental results show that the proposed implementation achieves higher or the same frame rates as the previously designed systems, but at a much higher, HD resolution. Furthermore, superiority of the design over GPU solutions, under similar benchmark tests, is shown in the comparison.

6. REFERENCES

[1] R. Szeliski and H-Y. Shum, "Creating Full View Panoramic Image Mosaics and Environment Maps," in *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1997, SIGGRAPH '97, pp. 251–258, ACM.

[2] S. E. Chen, "Quicktime VR: An Image-based Approach to Virtual Environment Navigation," in *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA, 1995, SIGGRAPH '95, pp. 29–38, ACM.

[3] M. Brown and D. Lowe, "Automatic Panoramic Image Stitching Using Invariant Features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, August 2007.

[4] S. Peleg and J. Herman, "Panoramic Mosaics by Manifold Projection," in *IEEE Conference on Computer Vision and Pattern Recognition*, San Juan, Puerto Rico, June 1997, pp. 338 –343.

[5] P. Burt and E. Adelson, "A Multiresolution Spline with Application to Image Mosaics," *ACM Trans. Graph.*, vol. 2, no. 4, pp. 217–236, Oct. 1983.

[6] M-S. Su, W-L. Hwang, and K-Y. Cheng, "Variational Calculus Approach to Multiresolution Image Mosaic," in *Proceedings*

of International Conference on Image Processing, Oct. 2001, vol. 2, pp. 245 –248.

[7] B. Wilburn, N. Joshi, V. Vaish, E-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy, "High Performance Imaging Using Large Camera Arrays," *ACM Trans. Graph.*, vol. 24, pp. 765–776, July 2005.

[8] H. Afshari, A. Akin, V. Popovic, A. Schmid, and Y. Leblebici, "Real-Time FPGA Implementation of Linear Blending Vision Reconstruction Algorithm Using a Spherical Light Field Camera," in *IEEE Workshop on Signal Processing Systems*, 2012.

[9] B. Daga, A. Bhute, and A. Ghatol, "Implementation of Parallel Image Processing Using NVIDIA GPU Framework," in *Proceedings of International Conference on Advances in Computing, Communication and Control*, Mumbai, India, January 2011, pp. 457–464.

[10] P. P. Shete, P. P. K. Venkat, and S. K. Bose, "Pyramidal Image Blending Using CUDA Framework," *International Journal of Engineering Science and Technology*, vol. 3, no. 12, pp. 8502–8513, December 2011.

[11] O. Sims and J. Irvine, "An FPGA Implementation of Pattern-Selective Pyramidal Image Fusion," in *International Conference on Field Programmable Logic and Applications*, August 2006, pp. 1–4.

[12] Y. Song, K. Gao, G. Ni, and R. Lu, "Implementation of real-time Laplacian pyramid image fusion processing based on FPGA," *Proceedings of SPIE*, vol. 6833, 2007.

[13] G. S. Van Der Wal and P. J. Burt, "A VLSI Pyramid Chip for Multiresolution Image Analysis," *International Journal of Computer Vision*, vol. 8, no. 3, pp. 177–189, Sept. 1992.

[14] R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, New York, NY, USA, 2011.

Original article published in IEEE Transactions on Circuits and Systems II: Express Briefs, 61(10):803-807, October, 2014

Performance Optimization and FPGA Implementation of Real-Time Tone Mapping

Vladan Popovic, *Student Member, IEEE,* Elieva Pignat, and Yusuf Leblebici, *Fellow, IEEE*

***Abstract*—This brief analyzes the performance of the hardware-based tone mapping operators for compression of high dynamic range images. The bottlenecks of a tone mapping system are determined and a high-performance field programmable gate array (FPGA) implementation of an operator is introduced. The operator utilizes polynomial mapping technique, adaptive to the pixel values; hence preserving high contrast areas. The technique is further optimized for the presented resource-efficient FPGA implementation. We show that the timing optimization does not reduce the image quality, by obtaining high peak signal-to-noise-ratio of the resulting images. The timing comparison to the similar implementations shows 2.5 times increase in the achieved throughput, irrespective of the hardware platform.**

I. Introduction

Modern digital cameras are still not able to capture the full dynamic range of natural scenes, *i.e.* the ratio between intensity of the brightest and the darkest pixel. This results in underexposed or overexposed regions of the taken image and the lack of local contrast. Fig. 1 shows three shots taken under different exposure settings of the camera. The underexposed and overexposed images show fine details in very bright and very dark areas, respectively. These details cannot be observed in the moderately exposed image.

The increase of dynamic range and contrast enhancement are only a few of the methods used to create a realistic representation of a scene. High dynamic range (HDR) imaging technique was introduced to increase the dynamic range of a captured scene by encoding images with higher precision than standard 24-bit RGB. The most common method of obtaining HDR images is by taking several low dynamic range (LDR) photographs, all under different exposures. Such example is given in Fig. 1. Debevec and Malik [1] developed an algorithm for creating wide range radiance maps from multiple LDR images. The algorithm included obtaining camera response curve, creation of HDR radiance map and storage in RGBE format [2]. Alternatively, scenes can be represented using exposure fusion [3]. Exposure fusion is a pipelined process on LDR images and it is not necessary to create a large HDR image, which significantly reduces memory requirement. A similar principle is used for contrast enhancement using a single LDR image [4].

V. Popovic, E. Pignat and Y. Leblebici are with the Microelectronic Systems Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland. e-mail: vladan.popovic@epfl.ch.

This work has been partially funded by the Science and Technology Division of the Swiss Federal Competence Center Armasuisse.

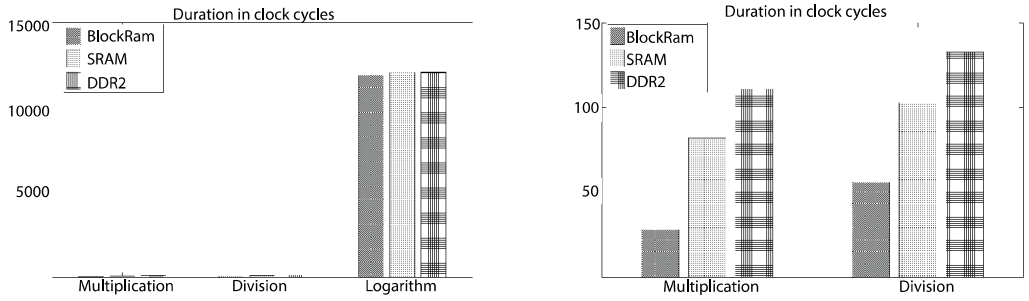
(a) (b) (c)

Fig. 1. Stanford Memorial church photographed with: (a) short, (b) medium and (c) long exposure time. Courtesy of Paul Debevec.

Apart from capturing natural scenes, another problem arises when displaying them. Current displays are limited to a very small dynamic range, and suffer from problems of displaying even standard LDR images. Thus, tone mapping operation [5] is introduced to map the real pixel values to the ones adapted to the displaying device. The purpose of tone mapping is to reduce the contrast in the HDR image, while preserving natural features of the scene. Due to characteristics of the human visual system (HVS), it is enough to tone map only the luminance component of the pixels.

Tone mapping operators can be divided into two main groups named global and local operators. Global operators are spatially invariant because they apply the same transformation to each pixel in the image. These algorithms usually have low complexity and high computational speed. One of the first complex global techniques was introduced by Ward et al. [5], which included histogram equalization and sensitivity of HVS to contrast in the image. Later, Pattanaik et al. [6] created a time-dependent operator based on HVS model and subjective experiments. The operator also takes into account color appearance and changes of color perception over time. One of the latest global techniques is based on adaptive logarithmic mapping and it was introduced by Drago et al. [7]. Even though it is considered as a global technique, it applies different mapping curves on pixels based on their luminosity. The curves vary from $\log_2$ for the darkest pixels to $\log_{10}$ for the brightest.

However, global operators do not preserve local contrast in the images where the luminance is uniformly occupying the full dynamic range. Oppositely, local operators are more flexible and adaptable to the image content, which may drastically improve local contrast in regions of interest [8]–[10]. Since they differently operate on different regions of the



(a) Complete benchmark results

(b) Magnified results for short operations

Fig. 2. Performance measurement results of the benchmark system. Duration is measured in clock cycles for three mathematical operations and three data storage mediums. The results show that (a) logarithm is the most process-intensive operation, and that (b) the relative influence of the storage medium is higher in faster operations.

image, they are significantly more expensive and resource-demanding. Hassan and Carletta [11] proposed an FPGA architecture for Reinhard operator [12], which introduces a local adaptation inspired by photographic film development in order to avoid halo artifacts. However, the proposed local adaptation requires a Gaussian pyramid decomposition, which requires a vast amount of resources, especially in terms of the required memory. Additionally, Hassan and Carletta [11] propose to estimate the logarithm function by the integer part of the operand and then refine it using the look-up table (LUT) with a fixed number of bits. A large number of bits should be considered in order to preserve the dynamic range, which enlarges the LUT size even further. A method with a lower number of bits reduces the precision and the dynamic range of the image.

Lapray et al. [13] presented a full imaging system with an FPGA as a processing core. Even though the used camera sensor streams 60 fps signal, the system provides only 30 fps output video for 1 Mpixel frame. The loss of frame rate is due to a calibration step needed before tone mapping and a slow non-pipelined implementation of computations, such as division and logarithm. The computational results are precise, but the cost is significantly reduced frame rate. Apart from FPGA systems, a system-on-chip (SOC) solution was presented by Chiu et al. [14]. ARM SOC platform was used for both global and local tone mapping processor. The achieved frame rate was 60 fps for a 1024 $\times$ 768 pixels resolution frame.

A detailed comparison of the major tone mapping operators is published by Yoshida et al. [15]. The comparison was realized by human subjects grading several aspects of the constructed image, such as contrast, brightness, naturalness and detail reproduction. One of the best graded techniques in this review was the global operator by Drago et al. [7]. Therefore, this operator is taken as a base for development of the FPGA-suitable operator.

In this work, we analyze performance of the tone mapping procedure on 1080p60 (1920×1080) input and output images and determine the logarithm calculation as the main bottleneck of the system. We constrain (1) the image quality to be at least 8 bits per color channel; (2) the system frame rate to be higher than 60 fps, and we aim to find a function that satisfies both constraints. We propose a polynomial approximation of the Drago operator, which overcomes the system's performance bottleneck, and show that this tone mapping function fits our constraints. An efficient FPGA implementation of the operator is presented together with the supporting prototype system. Our implementation exploits pipeline processing of the polynomial approximations, which increases performance compared to the previously used LUT-based methods. Furthermore, performance and image quality measurements show a significant increase in the achieved frame rate, without any visually perceivable loss of quality in the image.

II. Problem Analysis

Similarly to the majority of the global operators, Drago operator uses logarithmic mapping function expressed in (1), where displayed luminance L_d is derived from the ratio of world luminance L_w and its maximum L_{max}. The algorithm adapts the mapping function by changing the logarithm base t as a function of the bias parameter b [7], as shown in (2). The base value is a function of the pixel luminance, and it is bounded on the interval $[2, 10]$.

$$L_d = \frac{\log_t(L_w + 1)}{\log_{10}(L_{max} + 1)} \tag{1}$$

$$t(b) = 2 + 8 \cdot \left(\frac{L_w}{L_{max}}\right)^{\frac{\ln b}{\ln 0.5}} \tag{2}$$

Even though this mapping is created for interactive applications, its speed is very low for video applications. The reported frame rate is below 10 frames per second (fps), even for 720 $\times$ 480 pixels image, without any approximations that decrease the image quality [7].

In order to analyze complexity and timing performance of the tone mapping algorithm, we implemented a benchmark system on Xilinx ML501 Development Kit, which includes XC5VLX50 Virtex-5 FPGA, and both Zero-bus turnaround (ZBT) SRAM and DDR2 memory chips. The benchmark system included a MicroBlaze microprocessor, memory controllers for code and data storage, DVI (Digital Visual Interface) controller for display of the tone mapped image, and

timer for measuring performance. The tone mapping code was written in C and it was run on MicroBlaze.

The purpose of this benchmark is to determine the bottleneck of the system in terms of timing performance. We wanted to determine what is the most time-consuming operation, and what is the influence of the external memory on performance of this FPGA system. We benchmarked multiplication, division and logarith, since they appear in Equations (1) and (2). Furthermore, operand data are stored in three different locations: internal BlockRAMs (BRAM), external SRAM, and external DDR2 memory. Hence, nine different cases were observed and measured. The instantiated timer measured the number of clock cycles needed for each operation, in each case. Thus, a performance analysis is independent of the clock frequency. The numbers are obtained by averaging 10000 measurements per each case.

Fig. 2(a) shows that the most time-consuming operation is logarithm, which is the bottleneck of the system. Furthermore, it shows that the relative improvement when the fast memories are used is negligible. Fig. 2(b) illustrates in more detail the duration of multiplication and division, which is two orders of magnitude lower than the duration of logarithm calculation. The dominant factor in these two cases is the memory access, as the duration can even triple when external DDR2 memory is used.

The analysis showed two main issues in tone mapping operation: (1) Fast calculations are significantly affected by external memory access, and (2) Long duration of the logarithm operation. In order to resolve these issues, we implemented the whole tone mapping as a hardware-only system. The system resembles an accelerating unit, with reduced access to the external memory and faster logarithm calculation.

III. Tone Mapping Optimization

The dynamic range of the natural scenes reaches values as high as 400'000, according to [1] and [7]. Although an exact representation requires 19 bits, modern displays using DVI standard do not support more than 8 bits per color. We decided to use 16-bit calculations in order to achieve computational precision higher than 8 bits, with large enough error margin.

To avoid long calculation time and large LUTs, we have developed an optimized operator based on (1) and (2). Using the logarithm properties, we can change its base and calculate only natural and base-10 logarithms. Expressions in the form of $\log(1+x)$ can be efficiently approximated by the Chebyshev polynomials of the first kind $T_i(x)$ [16]. The approximation needs only six integer coefficients to achieve the desired 16-bit precision. The intermediate approximation step is given in (3), where L_{wa} is the world adaption luminance calculated as the log-average of all pixels' luminance.

$$L_d = \frac{\sum_{i=0}^{5} c_e(i) T_i\left(\frac{L_w}{L_{wa}}\right)}{\log_{10}\left(\frac{L_{max}}{L_{wa}}+1\right)\cdot \ln\left(2+8\left(\frac{L_w}{L_{max}}\right)^{\frac{\ln b}{\ln 0.5}}\right)} \quad (3)$$

The Chebyshev approximation can be applied to both natural and base-10 logarithm by only changing the coefficients. The

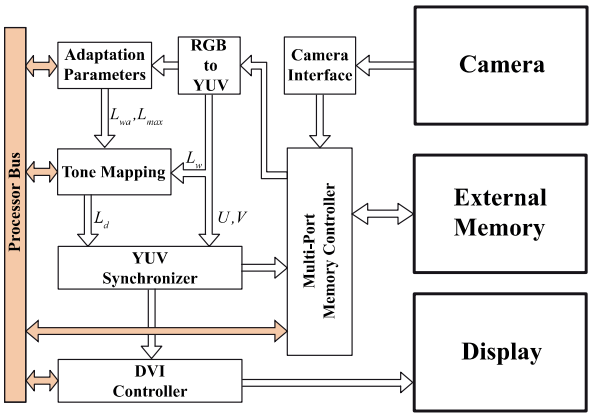


Fig. 3. Top-level architecture of the implemented system. Camera, memory and display are external to the FPGA. Internal architecture of the tone mapping and adaptation parameters blocks is shown in Fig. 4. MicroBlaze is connected as a master to the Processor Bus, but it is not shown in the diagram.

coefficients for the natural logarithm are denoted as $c_e(i)$, while $c_{10}(i)$ are for base-10 in (4). According to [7], the best visually perceived results are obtained for the bias parameter $b \approx 0.85$. We have fixed this parameter to $b = 0.84$, to simplify the hardware implementation, as generic power functions are difficult to implement. The exponent in the denominator becomes 0.25 and the argument can be evaluated by two consecutive calculations of the square root. The square root is also approximated using the Chebyshev polynomials.

$$L_d = \frac{\sum_{i=0}^{5} c_e(i) T_i\left(\frac{L_w}{L_{wa}}\right)}{\sum_{i=0}^{5} c_{10}(i) T_i\left(\frac{L_{max}}{L_{wa}}\right)\cdot \ln\left(2+8\left(\frac{L_w}{L_{max}}\right)^{\frac{1}{4}}\right)} \quad (4)$$

The natural logarithm term in the denominator cannot be precisely approximated by Chebyshev polynomial, due the arguments much higher than 1. A suitable approximation of the expression $\ln x$ is a fast convergence form of the Taylor series, which is expressed in (5). This expression needs only three non-zero coefficients to achieve a sufficient 16-bit precision, but the argument should be preconditioned as shown in (5). The world adaptation luminance L_{wa} is calculated as the log-luminance average of all N pixels. Thus, it can be calculated using Taylor approximation:

$$\ln x = 2\sum_{k=1}^{3} \frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1} \quad (5)$$

$$L_{wa} = \frac{1}{N}\sum_{N} \ln(L_w) \quad (6)$$

The equations (4)-(5) describe the optimized tone mapping operator suitable for hardware implementation. The set of required mathematical operations is reduced to only addition, multiplication and division.

IV. FPGA Implementation

The internal FPGA architecture of the system is shown in Fig. 3. Camera block represents an acquisition device, which

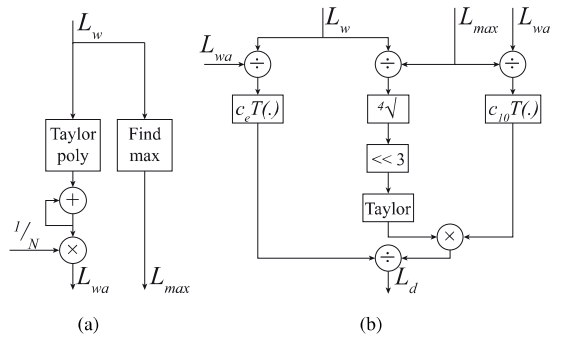


Fig. 4. Internal architecture of (a) adaptation parameters calculator, (b) tone mapping operator. Taylor and Chebyshev polynomials are evaluated using pipelined Horner scheme.

can be a single HDR sensor streaming video signal [13] or a single LDR camera taking multiple shots under different exposure settings. In both cases, the Camera Interface outputs HDR images that are written to the memory. Since the main goal of this work is to implement a tone mapping operator, the camera block is emulated and it streams a pre-calculated HDR radiance map [1] and stores it in DDR2. The pixel values are read from DDR2 and transformed into YUV color system.

The tone mapping implementation consists of two parts: finding the adaptation parameters (L_{max} and L_{wa}) and tone mapping curve implementation. L_{max} and L_{wa} must be calculated before starting the core tone mapping operation. Since the presented system processes the HDR video stream, the parameters are determined based on the previous frame, under the assumption that the scene illumination does not vary faster than response time of the HVS. The parameters are updated at the end of each frame.

The block diagram of a subsystem for finding the adaptation parameters is shown in Fig. 4(a). The problem of finding L_{max} consists of finding the maximum value in a sequence of read luminances. The world adaptation luminance is calculated by accumulating Taylor approximation evaluations and averaging them by the total number of pixels in the frame.

Fig 4(b) presents the block diagram of the tone mapping function. The Chebyshev polynomials approximating logarithm and square root are evaluated using the pipelined Horner scheme, and the division block is implemented using the fast Anderson algorithm [16]. The Taylor approximation blocks in Fig. 4(a) and Fig. 4(b) follow the implementation given in Algorithm 1. Taylor approximation of the logarithm given in (5) is accurate only in the range $[0, 1]$. However, logarithm argument in the denominator of expression (4) is in the range $[2, 10]$. Thus, the argument is scaled down to the $[0, 1]$ range. The scaling factor is determined by the location of the first "1" in the 16-bit fixed-point representation. Additionally, the argument is preconditioned according to (5) in order to achieve fast convergence. The polynomial is evaluated using the pipelined implementation of the Horner scheme. The scheme requires both zero and non-zero coefficients $q(k)$ to be provided; hence, the processing pipeline of (5) comprises six stages instead of three. The polynomial is scaled back to the original range after the evaluation.

Algorithm 1 Taylor polynomial of $\ln x$

1: $y := position\_of\_the\_first\_1(x)$
2: $x_{scaled} := x \,/\, 2^y$ { % Scale to [0,1] range}
3: $t := \frac{x_{scaled}-1}{x_{scaled}+1}$ {% Precondition for fast convergence}
4: {% Horner scheme}
5: $temp(6) := 0$
6: **for** $k = 5$ downto 0 **do**
7: $\quad poly(k) := temp(k+1) + q(k)$
8: $\quad temp(k) := poly(k) \cdot t$
9: **end for**
10: $result := poly(0) + y \cdot \ln 2$ {% Bring to original range}

The output L_d of the tone mapping block is synchronized with the corresponding chrominance values in the YUV Synchronizer. The tone mapped image can be directly shown on a screen, or written back to the memory.

V. Experimental Results

The tone mapping system was implemented on ML501 Development Kit with Xilinx XC5VLX50FFG676 Virtex-5 FPGA. The maximum operational frequency reported by the synthesis tool is 214.27 MHz and the utilization summary is given in Table I. The summary presents the utilization of a single tone mapping block, and the full system which includes DDR2 and DVI controllers. As a comparison, the utilization summary from [13] is also given, since the same FPGA family is used. Our implementation instantiates more DSP blocks, which are used for multiplication, whereas utilization of all the other resources is significantly reduced.

The comparison of the proposed implementation in terms of achievable pixel throughput is given in Table II. Comparison data are calculated from the reported values in original publications. The systems in the comparison were developed on different platforms. In order to have a fair comparison, we additionally synthesized our design for Stratix-II and for UMC 180 nm technology node, and obtained throughput results from post-place-and-route simulations. The comparison results show approximately 2.5 times improvement over the currently best implementation. The advantage of our implementation is

TABLE I
FPGA device utilization summary

Resources	Tone Mapping	Full System	[13]
Slice LUTs	918	4536	14168
Slice Registers	540	5036	8132
BlockRAM/FIFO	0	8	40
DSP48Es	30	30	4

TABLE II
Pixel throughput comparison. Values are in Mpixels/s

Platform / Node	GPU	Virtex-5		Stratix-II		UMC 180	TSMC 130
System	[7]	Our	[13]	Our	[11]	Our	[14]
Throughput	13	133	39	111	47	124	47

(a) (b)

Fig. 5. Examples of the tone mapped HDR images using (a) our fast implementation, and (b) Drago operator [7]. The reduction in quality is not noticeable in the images when the fast implementation is used. Radiance maps courtesy of Paul Debevec and Raanan Fattal.

TABLE III
HDR IMAGE QUALITY MEASUREMENTS

	House	*Synagogue*	*Cathedral*	*Memorial*
PSNR [dB]	59.7	73.46	60.79	74.15
SSIM	0.9990	0.9999	0.9996	0.9995

the fully pipelined operation, which reduces the critical path of the system. Furthermore, our implementation allows larger image resolutions if a lower frame rate is acceptable.

Visual quality testing was applied on a set of HDR images provided by Debevec [1] and Fattal [9]. These images were stored into DDR2 and loaded twice by the tone mapping block. Maximum and log-average luminance are calculated in the first "pass", and the tone mapping is realized after. Screenshots of four tone mapped examples are shown in Fig. 5(a). The results show that the proposed approximations and hardware implementation do not introduce visual differences compared to the results of [7] which are shown in Fig. 5(b).

Apart from visual appearance comparison, an objective image quality comparison was performed. Peak signal-to-noise-ratio (PSNR) and Structural Similarity Index (SSIM) [17] values are calculated and shown in Table III. The tone mapped images in Fig. 5(b) are taken as the reference images. SSIM values almost equal to 1 confirm that the structure of the scene's objects is not affected. High PSNR values show that the objective image quality is high, despite lower computational time when the proposed fast implementation is used. Moreover, the PSNR is high enough that the difference in images cannot be visually noticed on the standard LDR screens.

VI. Conclusion

In this paper, we proposed an optimized global tone mapping operator based on Drago operator. The operator is suitable for high frame rate operation and the presented resource-efficient FPGA implementation. The operator compresses luminance component of the HDR image using Taylor and Chebyshev polynomials, since the number of needed coefficients for high precision calculation is low. Horner scheme is used for evaluation of the polynomials and it is implemented as a fully pipelined operation, resulting in high performance. High operating frequency and increased frame rate make this algorithm and implementation an excellent choice for real-time and video HDR applications.

References

[1] P. E. Debevec and J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," in *ACM SIGGRAPH 97*, New York, NY, USA, 1997, pp. 369–378.

[2] G. Ward, *Graphics Gems II*. San Diego, CA, USA: Academic Press, 1991, ch. Real Pixels, pp. 80–83.

[3] T. Mertens, J. Kautz, and F. Van Reeth, "Exposure Fusion," in *Pacific Conf. on Computer Graphics and Applications*, 2007, pp. 382–390.

[4] A. Saleem, A. Beghdadi, and B. Boashash, "Image fusion-based contrast enhancement," *EURASIP Journal on Image and Video Processing*, vol. 2012, no. 10, 2012.

[5] G. Ward, H. Rushmeier, and C. Piatko, "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Trans. Vis. Comput. Graphics*, vol. 3, no. 4, pp. 291–306, Oct. 1997.

[6] S. N. Pattanaik, J. Tumblin, H. Yee, and D. P. Greenberg, "Time-dependent visual adaptation for fast realistic image display," in *ACM SIGGRAPH 00*, New York, NY, USA, 2000, pp. 47–54.

[7] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, "Adaptive Logarithmic Mapping For Displaying High Contrast Scenes," *Computer Graphics Forum*, vol. 22, no. 3, pp. 419–426, 2003.

[8] F. Durand and J. Dorsey, "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002.

[9] R. Fattal, D. Lischinski, and M. Werman, "Gradient Domain High Dynamic Range Compression," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 249–256, Jul. 2002.

[10] R. Mantiuk, S. Daly, and L. Kerofsky, "Display adaptive tone mapping," *ACM Trans. Graph.*, vol. 27, no. 3, Aug. 2008.

[11] F. Hassan and J. E. Carletta, "An FPGA-based architecture for a local tone-mapping operator," *Journal of Real-Time Image Processing*, vol. 2, no. 4, pp. 293–308, 2007.

[12] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda, "Photographic Tone Reproduction for Digital Images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 267–276, Jul. 2002.

[13] P.-J. Lapray, B. Heyrman, M. Rosse, and D. Ginhac, "HDR-ARtiSt: High Dynamic Range Advanced Real-time Imaging System," in *IEEE Int. Symp. on Circuits and Systems*, 2012, pp. 1428–1431.

[14] C.-T. Chiu, T.-H. Wang, W.-M. Ke, C.-Y. Chuang, J.-R. Chen, R. Yang, and R.-S. Tsay, "Design optimization of a global/local tone mapping processor on arm SOC platform for real-time high dynamic range video," in *IEEE Int. Conf. on Image Processing*, 2008, pp. 1400–1403.

[15] A. Yoshida, V. Blanz, K. Myszkowski, and H.-P. Seidel, "Perceptual Evaluation of Tone Mapping Operators with Real-World Scenes," in *SPIE Human Vision & Electronic Imaging X*, 2005, pp. 192–203.

[16] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 3rd ed. Berlin, Germany: Springer-Verlag, 2007.

[17] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–312, Apr. 2004.

Original article published in Proceedings of IEEE International Conference on Very Large Scale Integration (VLSI-SOC), Playa Del Carmen, Mexico, 2014

Reconfigurable Forward Homography Estimation System for Real-Time Applications

Vladan Popovic and Yusuf Leblebici
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland.
vladan.popovic@epfl.ch, yusuf.leblebici@epfl.ch

***Abstract*—Image processing and computer vision algorithms extensively use projections, such as homography, as one of the processing steps. Systems for homography calculation usually observe homography as an inverse problem and provide an exact solution. However, the systems processing larger resolution images cannot meet inherently tight real-time constraints. Look-up table based systems provide an option for forward homography solutions, but they require large memory availability. Recent compressed look-up table methods reduce the memory requirements at the expense of lower peak signal-to-noise-ratio. In this work, we present a forward homography estimation algorithm which provides higher image quality than compressed look-up table methods. The algorithm is based on bounding the homography error, and neglecting the pixels out of the determined bound. The presented FPGA implementation of the estimation system requires a small amount of hardware, and no memory storage. The prototype system project an image frame onto a spherical surface at 295 Mpixels/s rate which is, up to our knowledge, currently the fastest homography system.**

I. Introduction

A homography, or projective transformation, is a mapping of vectors belonging to the same projective space. It is often used in image processing and computer vision to change the perspective view of the image [1]–[3]. An illustration of a projective transformation is shown in Fig. 1(a). The planar view $\mathcal{I}$ of the scene is transformed into an alternate view $\mathcal{J}$. The point $\mathbf{x}$ in the scene is observed at different locations in two given views, which is illustrated in Fig. 1(b) on a real image example.

A homography can be defined for any N-dimensional projective space. However, the most common ones are two-dimensional (planar) and three-dimensional (spherical and cylindrical) projections. Besides changing the perspective view, planar homography is used for camera calibration [4], stereo camera rectification [5]–[8], and object tracking and focusing [9], [10]. Applications of 3D homography vary from estimation of 3D scene models [11] and volume [12] to generating cylindrical and spherical panoramas [13], [14].

In this work, we consider applications such as real-time spherical panorama construction. Real-time homography is usually perceived as an inverse problem thanks to rather simple reconstruction pipeline. Reconstructing panoramic views requires multiple shots that are stitched into one single image. Inverse homography is suitable for this application, since the raw images can be stored into memory before performing the actual reconstruction. For each desired pixel in the reconstruction, the most appropriate pixel can be found in the original images. The mapping function is either determined by using runtime calculations [14] or pre-calculated and stored in lookup tables (LUTs) [13]. Thus, inverse homography is determined by straightforward implementation of mathematical functions [14].

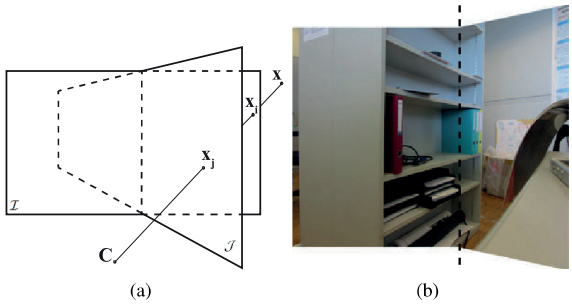


Fig. 1. a) Illustration of 2D homography on the example of two different views of the same scene point $\mathbf{x}$. Points $\mathbf{x_i}$ and $\mathbf{x_j}$ are related by homography; b) Homography example on a real photograph. The same photograph is projected on two different view planes. Intersection line of the two planes is marked by a dashed line. Change of perspective is noticed in the right part of the image.

However, timing constraints become very tight when the desired output resolution is high. The image processing systems can hardly meet the real-time constraints of 25-30 frames per second (fps) when reconstructing high resolution images. Forward homography is suggested as a possible solution to this problem in stereo image rectification systems [6]–[8], where the same real-time constraints apply.

The forward homography solves the issue of system constraints, such as memory bandwidth, since the correct destination is calculated for each input pixel. Hence, only the necessary pixels for the final reconstruction are stored in memory, thus reducing the required bandwidth. The destination coordinates are pre-calculated offline and stored in registers of the processing system. Another advantage of the LUT-based approach is that LUT size is independent of the final image resolution.

However, LUT size linearly increases with respect to the raw image resolution. Modern cameras have more than 10 Mpixels, and LUTs become too large for on-chip storage. Compressed LUT methods [8] may partially solve this problem, but the peak-signal-to-noise ratio (PSNR) drops signifi-

978-1-4799-6016-3/14/$31.00

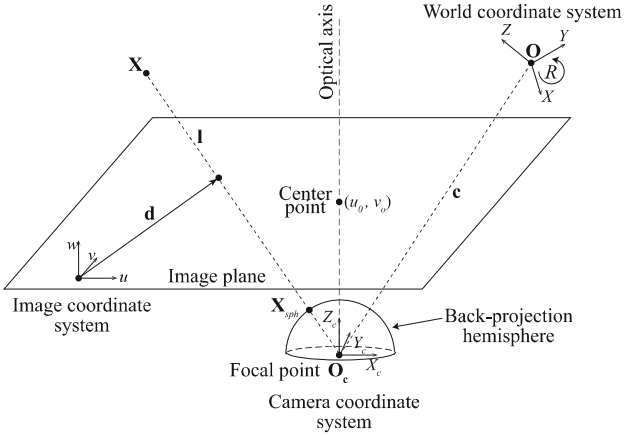


Fig. 2. Geometric transformations during image formation process. The world, camera, and image coordinate systems are shown with relations between them. A light ray passing through the world scene point $\mathbf{X}$ and the sensor's focal point intersects the image plane in point $\mathbf{d}$, which represents a pixel in the acquired image. The back-projection procedure reconstructs the original light ray $\mathbf{l}$ and locates the intersection point with the back-projection hemisphere, $\mathbf{X}_{sph}$.

cantly in the presence of large differences between input and output image resolution. These differences are observed in the majority of modern cameras, whose high resolution images are usually displayed on 2 Mpixels monitors. Analysis of this problem is given in Section III.

We propose a novel method for real-time forward homography based on direct calculation of destination pixels. The system operates on a pixel stream coming from a camera, and it is appropriate even for large ratios of input and output resolutions. We determine the optimal projection error bound based on camera calibration, and disregard all projections with the error higher than the determined bound. The chosen error bound provides the highest PSNR of the projected image. The algorithm is prototyped on an FPGA development system for the spherical panorama application. The prototype provides input/output pixel ratio of more than 200, *i.e.* more than 200 pixels are mapped into a single one.

II. Theoretical Background

The image formation can be approximated by the pinhole camera model for many computer vision applications [1]. The pinhole camera projects a 3D world scene into a 2D image plane. In order to perform this projection, three coordinate systems are considered, as depicted in Fig. 2. The coordinates of point $\mathbf{X}$ are expressed in the world coordinate system with its origin at the point $\mathbf{O}$. The same point can also be expressed in the camera coordinate system by coordinates $\mathbf{X_c}$. Origin of the camera coordinate system coincides with the camera's focal point $\mathbf{O_c} = \mathbf{C}$. The relation between the world and camera coordinates is unique and consists of a single translation and a single rotation:

$$\mathbf{X_c} = R(\mathbf{X} - \mathbf{c}) \tag{1}$$

The vector $\mathbf{c}$ denotes the distance between origins of the world and the camera coordinate systems, whereas the rotation matrix R expresses three Euler rotations in order to align the mentioned systems' axes. Parameters R and $\mathbf{c}$ are called extrinsic camera calibration parameters.

The image coordinate system, has aligned axes with the camera coordinate system. Position of the projected point in the image coordinate system is expressed by:

$$\mathbf{d} = M(\mathbf{X} - \mathbf{c}) = K\mathbf{X_c} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{X_c} \tag{2}$$

where $\mathbf{d}$ is the pixel position in the image coordinate system, f is the focal length, M is the projection matrix, and K is the intrinsic calibration matrix. The origin of the system is coincident with one of the corners of the image sensor. Thus, an additional shift of the sensor's central point to the sensor's corner is needed to obtain the final pixel position.

III. Estimation Algorithm

Estimating a forward homography in real-time is not a trivial problem. The system should determine the final pixel position, *e.g.* position in a panoramic image, based only on the pixel coordinates in the original frame. The problem arises due to non-integer values of the mapped pixel coordinates (see Fig. 4). When observing homography as an inverse problem, it is easy to scan through the desired pixel grid and choose the closest pixel from the original frame. Forward homography processes a pixel stream, and the system can determine the closest position on the destination pixel grid. However, it cannot determine if the current pixel in the stream is the best option. Thus, pixels that are mapped to the same position are overwritten and the last pixel that appears in the stream will be considered as the correct one. Hence, the PSNR is significantly decreased. This problem is even more emphasized in modern high-resolution cameras when hundreds of pixels are mapped into a single one, which is shown in the results section.

We developed a new homography estimation algorithm to overcome this issue. Equation (2) expresses the projection of a point in the 3D space onto the image plane. The first step of the algorithm is to back-project the pixels from the image frame. Each pixel $\mathbf{d}$ is back-projected into a line $\mathbf{l}$ in a 3D world that includes the focal point (projection center) $\mathbf{O_c}$. The line is illustrated in Fig. 2 and expressed by the inverse of Equation (2):

$$\mathbf{l} = M^{+} \begin{bmatrix} \mathbf{d} & 1 \end{bmatrix}^{\top} \tag{3}$$

where M^{+} denotes a Moore-Penrose pseudoinverse of the projection matrix. Thus, back-projection results in a set of lines (light rays), where each of them contains the focal point of the lens. In order to obtain 3D world coordinates, we should define a back-projection surface. We choose a unit sphere, $|r| = 1$, for the purpose of panorama construction. Other options include a cylindrical projection, or a planar projection for image rectification or disparity estimation applications. Back-projection onto the unit sphere is performed by normalizing the line $\mathbf{l}$ by its L^2 norm, and transforming Cartesian (x, y, z) coordinates into spherical (θ, ϕ, r), where θ is the polar angle, ϕ is the azimuth, and r is the radius:

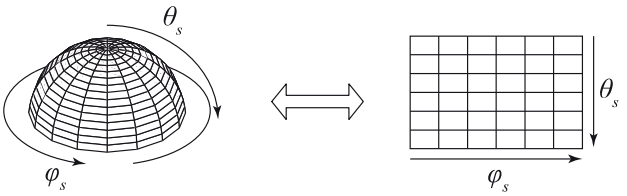


Fig. 3. The pixelized hemisphere on which the pixels are back-projected is shown on the left. The hemisphere can be unwrapped into an equivalent planar image shown on the right. The back-projection problem is regarded as homography between the original image coordinate system (source frame) and the unwrapped hemisphere (destination frame).

$$\begin{aligned} \mathbf{X}_{sph} &= \mathbf{l} / ||\mathbf{l}||_2 \\ \theta &= \arccos\left(\mathbf{X}_{sph}(z)\right) \\ \phi &= \arctan\left(\mathbf{X}_{sph}(y)/\mathbf{X}_{sph}(x)\right) \\ |r| &= 1 \end{aligned} \tag{4}$$

The spherical pixel grid is defined by the equidistant angles $(\theta_s, \phi_s, 1)$, where each pixel corresponds to a single angle pair. Fig. 3 shows a pixelized sphere and its unwrapped, planar representation.

The camera projection matrix M is obtained by camera calibration and the back-projection pixel grid (θ_s, ϕ_s) is defined *a priori*. If $\mathbf{X}_{sph}$ is the back-projected pixel, and $\mathbf{X}_s$ is the desired pixel on the hemispherical pixel grid, we define a projections error as:

$$e = ||\mathbf{X}_{sph} - \mathbf{X}_s||_2 \tag{5}$$

Afterwards, we find a threshold value ϵ, such that at least one distinct back-projected pixel $\mathbf{X}_{sph}$ exists for each grid pixel $\mathbf{X}_s$ with the error $e \leq \epsilon \leq \frac{\Delta}{\sqrt{2}}$, where Δ is the distance between two pixels on the hemisphere.

Five outcomes are possible in a 2D homography depending on the source and destination pixel positions. The simplest one is a 1-to-1 mapping when each pixel from the source frame maps to one in the destination frame. Furthermore, 1-to-0 and 0-to-1 are also possible, when the source pixel does not have a corresponding pixel in the destination frame, and vice versa. These three mappings are trivial cases and they will not be considered in the analysis. Complex 1-to-N and N-to-1 mappings occur when destination and source frames are oversampled, respectively. A possible real-time solution for 1-to-N mapping is suggested in [8]. However, a real-time solution with high PSNR has not yet been presented for N-to-1 mapping.

We resolve the 1-to-N mapping in the estimation algorithm by choosing the optimal ϵ. The optimal ϵ value ensures a distinct source pixel for each ϵ-neighborhood in the destination frame. Hence, a 1-to-N mapping is replaced by N 1-to-1 mappings.

Oppositely, ϵ value should be kept as low as possible in order to efficiently resolve N-to-1 mappings. The problem that arises in forward homography is that the N^{th} pixel in the stream is considered as the correct, unless a full mapping is

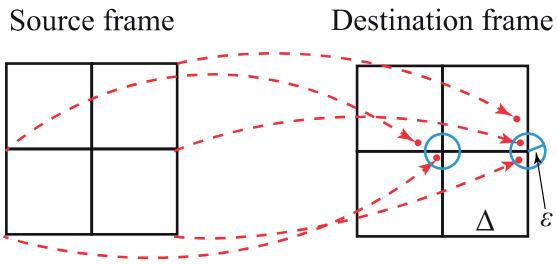


Fig. 4. Possible results of the proposed forward homography estimator. Intersections of black lines represent source and destination grid pixels, whereas red lines and dots are projections and projected pixel locations. The blue circles of radius ϵ mark the area in which the projected pixels are considered correct. When more than one pixel is within the blue circle, value of the last pixel in the incoming pixel stream is assigned to the pixel in the circle's center.

stored in internal LUT or external memory. Thus, the optimal ϵ is the lowest value that ensures 1-N resolving. Reduction of the ϵ value in the proposed estimation, also reduces the number of pixel candidates to $M < N$. The benefit of this reduction is two-folded: 1) increased chance of choosing the optimal pixel, and 2) lower error and higher PSNR when non-optimal pixel is chosen.

The homography between the image frame and unwrapped hemispherical surface is shown in Fig. 4. Intersections of black lines represent pixel positions on the respective grids. Red dashed lines illustrate homography between two frames, and red dots are projected pixel positions in the destination frame. Distances between red points and the closest intersection of black lines is the corresponding error e. The blue circles mark the ϵ-neighborhood in which projected pixels are considered as potential candidates for the final pixel value.

Fig. 4 illustrates two different cases of N-to-1 homography. Two source pixels on the left are mapped to the vicinity of a single destination pixel. The ϵ-neighborhood around the central destination pixel is set such that only one of the mapped pixels is inside the circle. Hence, the central destination pixel is given the value of the pixel inside the circle, which is indeed the closest projected pixel. In another situation, three pixels on the right side of the source frame are projected around one destination pixel. Two projections are inside the ϵ-neighborhood and one of them will be chosen as the destination pixel, *i.e.* the last one read out from the sensor.

IV. FPGA Implementation

An FPGA-based system is developed for the purpose of prototyping the real-time forward homography estimation. The full system architecture is shown in Fig. 5. The camera provides pixel values p in the raster scan order, *i.e.* line-by-line. Image is acquired using a single CMV20000 sensor, which outputs 20 Mpixels frames at maximum 30 frames per second (fps) rate. Pixels are streamed out of the camera as serialized 12-bit raw Bayer data. The Camera Interface block in Fig. 5 deserializes the pixel values and performs image pre-processing operations, such as white balancing and demosaicing. Additionally, an internal counter generates horizontal and vertical coordinates (u, v) of the processed pixel in the image frame (Fig. 2).

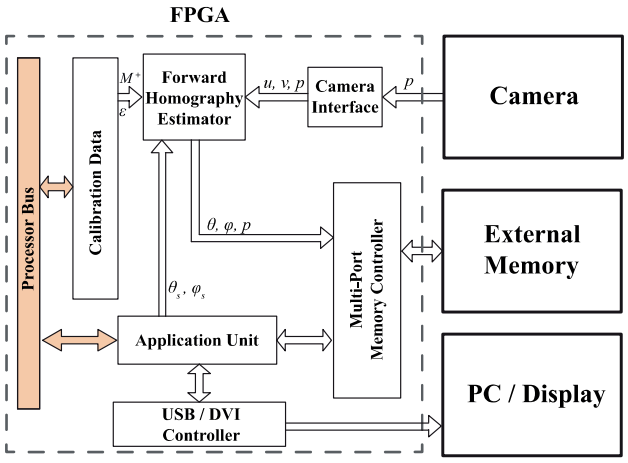


Fig. 5. Top-level FPGA architecture of the implemented system with the external peripherals. Dataflow is denoted by arrow directions.

Calibration Data block contains a bank of software accessible registers, and it is accessed by a MicroBlaze softcore microprocessor. The data stored in registers is obtained through the processes of intrinsic and extrinsic camera calibration and projection matrix calculation. The calibration data comprises the lens focal length f, sensor's central point position (u_0, v_0), the rotation matrix R, and the translation vector $\mathbf{c}$. These four parameters are used to calculate the projection matrix M and its pseudo-inverse. Only the elements of the pseudo-inverse matrix M^+ are stored in the registers of the Calibration Data block, since they are required for forward homography estimation.

The real-time image processing is realized inside the Application Unit. We implement the spherical panorama projection, which provides pixel positions on the spherical grid (θ_s, ϕ_s) to the Forward Homography Estimator (FHE). Since only one camera is connected in the current implementation, the final panorama results in a reduced field-of-view.

Internal architecture of the Forward Homography Estimator is shown in Fig. 6. Subtraction of the camera center point position translates the image frame origin to the frame center. Different row vectors of the matrix M^+ are provided to the dot product blocks, that evaluate the matrix multiplication in (3). A single dot product block in Fig. 6 is implemented as a pipelined multiply-accumulate unit in order to increase the speed performance of the system.

Equation (4) expresses the hemispherical back-projection and coordinate system change from Cartesian to spherical. The L^2 norm sub-block in Fig. 6 consists of two consecutive square root calculations. The square root module implements a CORDIC algorithm in the vectoring mode, which calculates the L^2 norm of its two inputs, *i.e.* $\sqrt{a^2 + b^2}$. The dividers for coordinate normalization are implemented using the iterative fast Anderson algorithm [15]. The transformation of coordinate system requires evaluation of inverse trigonometrical functions $\arctan$ and $\arccos$. The spherical angles (θ, ϕ) are calculated by applying the CORDIC algorithm to the Cartesian coordinates, as illustrated in Fig. 6.

The Application Unit provides information on the desired pixel grid . The error e from (5) is evaluated by the identical square root module used for the previous L^2 norm calculations. The error is compared to the pre-calculated ϵ, which is provided by MicroBlaze through a software accessible register. Output of the comparator serves as the output enable signal for a set of output registers, and as a write enable signal for the Multi-Port Memory Controller.

The FHE is a fully pipelined block. Each computation is followed by a register to shorten the critical path and increase the maximum frequency. Furthermore, each sub-block, *e.g.* dot product, square root, or trigonometric functions, is also pipelined providing a very fast operation of the system. The pipeline registers are not shown in Fig. 6.

A. Reconfigurability

The presented system is highly reconfigurable and has five degrees of freedom: camera resolution and frame rate, display frame resolution, projection surface, and the number of cameras. Fully pipelined processing allows change of camera resolution and frame rate even during the operation. The counter in the Camera Interface block uses horizontal and vertical synchronization signals from the camera to properly reset u and v values, hence it does not require resolution and frame rate information neither during synthesis, nor in run time. Modifying the display resolution or projection surface is performed in MicroBlaze and it does not influence the internal architecture of the FHE. MicroBlaze recalculates inverse M^+ of the projection matrix and the optimal ϵ, whereas operation of the FHE remains unchanged.

Additionally, the system can be reconfigured to support more than one camera. Multiple Camera Interface and FHE blocks can be instantiated and operated in parallel and independently. Thanks to the parallel operation, the maximum number of cameras is not constrained by system's performance, but by number of I/O pins and external memory bandwidth.

Furthermore, the proposed forward homography estimation is not dedicated to the developed prototype, and it can be used in any real-time system requiring image projections.

V. Experimental Results

The proposed FPGA design is implemented on a Xilinx Virtex-5 XUPV5-LX110T Evaluation Platform. A CMOSIS CMV20000 camera is mounted on a custom designed PCB, which is connected to the evaluation board. A photograph of the built setup is shown in Fig. 7. FPGA utilization summary of the FHE block and comparison with the designs that can perform the same function is given in Table I. The utilization reports were taken from the original publications. Even though these designs present their results for the image rectification applications, they are essentially implementations of forward homography. All of the compared works implement LUT-based methods, and hence require a lot of on-chip or off-chip memory. The proposed design uses less than 2% of the available LUTs and flip-flops on the used FPGA, and no internal nor external memory. Slightly higher utilization in terms of FPGA logic is due to real-time calculation of homography. The advantage of this design over the LUT-based

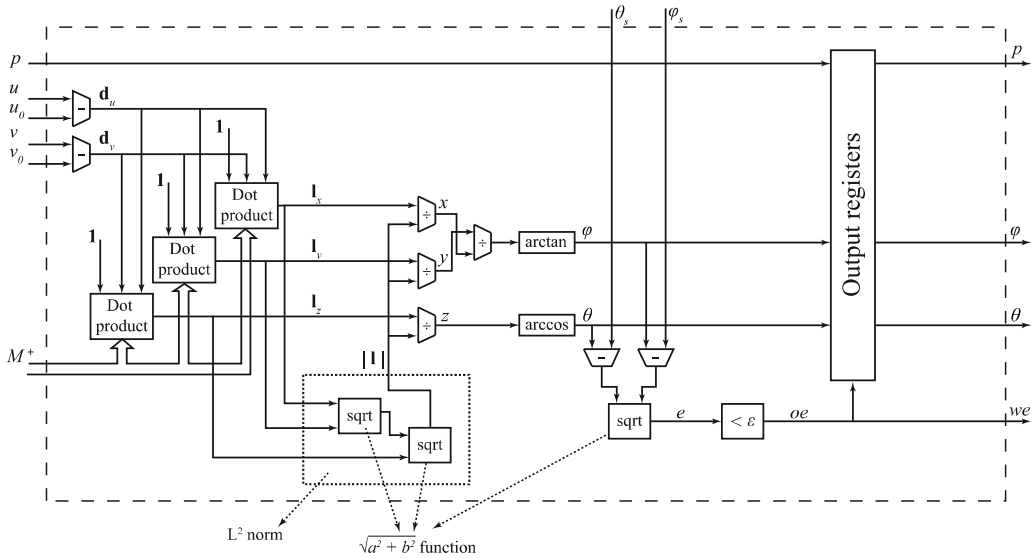


Fig. 6. Internal architecture of Forward Homography Estimator. The presented hardware evaluates expressions (3) – (5) using pipelined architecture. Pipeline registers are not shown for better visibility. The sub-blocks for square root and trigonometric functions evaluation utilize the CORDIC algorithm, whereas the fast Anderson algorithm [15] is used for implementation of the dividers.

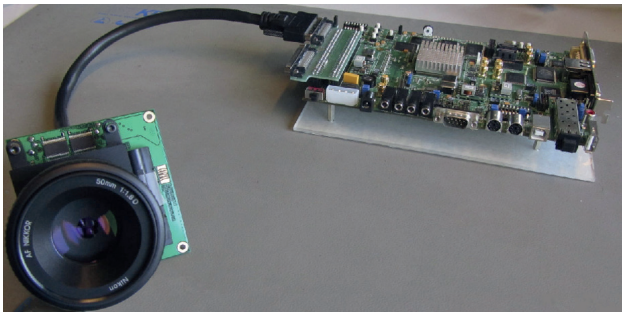


Fig. 7. Setup of the prototype system. A custom-made PCB with the installed image sensor is connected to the XUPV5 board with a VHDCI cable. A 50 mm Nikon F-mount lens is installed on top of the sensor.

TABLE I. FPGA DEVICE UTILIZATION SUMMARY AND COMPARISON

	FHE	[6]	[7]	[8]
Platform	Virtex-5	Virtex-5	Virtex-E	Virtex-5
Resolution	5120×3840	1280×720	640×512	1024×768
LUTs	1848	N/A	2459	784
Registers	1422	N/A	2075	427
BRAM [kB]	0	1300	99	104
DSP48Es	32	N/A	N/A	N/A

methods is that it can support very high image resolutions without any addition to the inferred hardware.

The measured maximum operating frequency of the implemented system is 308.16 MHz, which allows the system to process 15 fps of the full-resolution 20 Mpixels frames, *i.e.* 295 Mpixels/s. Furthermore, the input image resolution is reduced to 1024 × 768 for the purpose of comparison with

TABLE II. PSNR COMPARISON OF HOMOGRAPHY ESTIMATION

PSNR [dB]	Downsampled *Indoor*	*Indoor*	*Outdoor*
Resolution	1024×768	5120×3840	5120×3840
FHE	42.20	42.03	38.44
[8]	39.58	36.72	33.41

[8]. The measured frame rate is 391 fps, compared to 347 fps in [8].

The place and route tool reports a critical path through the large adders in the dot product evaluation sub-block. These adders utilize the FPGA logical elements, unlike multipliers that infer DSP48 blocks. Thus, it is possible to further increase the bandwidth by manually replacing the large adders with faster DSP48 blocks.

Results of the proposed forward homography estimation and its FPGA implementation are shown in Fig. 8(a) - 8(f). Fig. 8(a) and Fig. 8(d) represent an indoor and an outdoor scene at the input of the FHE system. Fig. 8(b) and Fig. 8(e) show the reduced field-of-view spherical panorama, at the output of the FHE system. Fig. 8(c) and Fig. 8(f) illustrate the unwrapped sphere suitable for display on 2 Mpixels screens.

The image quality loss is also evaluated for FHE and compared to the loss of LUT-based implementations. PSNR of the image in Fig. 8(c) and Fig. 8(f) are calculated, considering the inverse homography results as the ground truth images. Resolution of the reconstructed frame is 1920 × 1080, and region where the source frames are projected contains 87823 pixels.

Table II shows that the PSNR is increased compared to the LUT-based methods even in low-resolution projections, such as the downsampled *Indoor* image. Effects of the proposed ho-

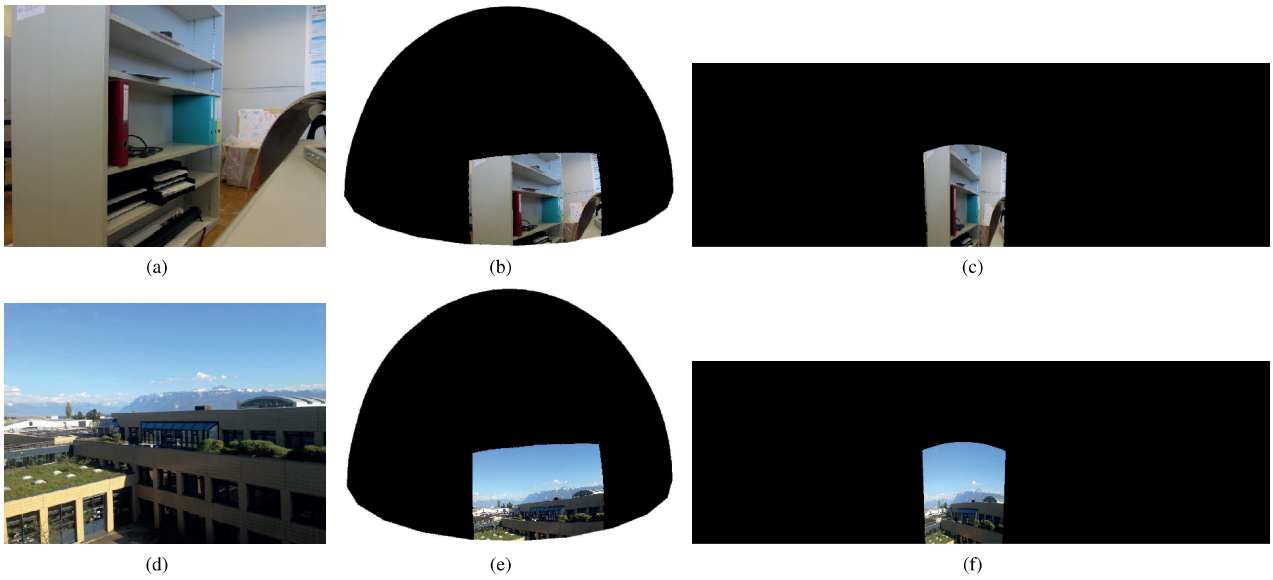


Fig. 8. Results of the proposed forward homography estimation: (a) and (d) The original 20 Mpixels *Indoor* and *Outdoor* images; (b) and (e) Outputs of the forward homography estimator displayed as true spherical projections; (c) and (f) unwrapped spherical projections displayed as 1920 × 1080 pixels panoramas.

mography estimation are emphasized with large input images. The PSNR of the full resolution *Indoor* image homography suffers a negligible drop, while the PSNR of the LUT-based method drops by 3 dB. The input image resolution does not influence the PSNR thanks to the flexibility of the ϵ parameter. The optimal error bound for the full resolution image is $\epsilon = 0.0061$, whereas $\epsilon = 0.11$ for the low resolution projection.

VI. Conclusion

In this manuscript, we presented a real-time forward homography estimation system. The proposed estimation is based on bounding the projection error, and reducing the number of incorrect pixels. The proposed FPGA implementation does not require internal LUTs or external memory for large projection data storage. The inferred hardware consumes a small amount of resources and provides real-time output, without significant loss in image quality. The presented estimation algorithm and its implementation outperform the state-of-the-art systems in terms of both system bandwidth and PSNR of the resulting image.

Acknowledgment

The authors gratefully acknowledge the support of Xilinx, Inc., through the Xilinx University Program. This work has been partially funded by the Science and Technology Division of the Swiss Federal Competence Center Armasuisse.

References

[1] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, 3rd ed. Cengage Learning, 2008.

[2] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[3] M. Sapienza, M. Hansard, and R. Horaud, "Real-time 3d reconstruction and fixation with an active binocular head," INRIA Grenoble, Tech. Rep. 7682, July 2011.

[4] Z. Chuan, T. D. Long, Z. Feng, and D. Z. Li, "A planar homography estimation method for camera calibration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, July 2003.

[5] R. Yang, M. Pollefeys, and S. Li, "Improved Real-Time Stereo on Commodity Graphics Hardware," in *IEEE Conference on Computer Vision and Pattern Recognition Workshop*, June 2004.

[6] D. H. Park, H. S. Ko, J. G. Kim, and J. D. Cho, "Real Time Rectification Using Differentially Encoded Lookup Table," in *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*. New York, NY, USA: ACM, 2011, pp. 47:1–47:4.

[7] C. Vancea and S. Nedevschi, "LUT-based Image Rectification Module Implemented in FPGA," in *IEEE International Conference on Intelligent Computer Communication and Processing*, 2007, pp. 147–154.

[8] A. Akin, I. Baz, L. Gaemperle, A. Schmid, and Y. Leblebici, "Compressed Look-Up-Table Based Real-Time Rectification Hardware," in *IFIP/IEEE 21st Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, Oct 2013, pp. 272–277.

[9] V. Vaish, G. Garg, E. Talvala, E. Antunez, B. Wilburn, M. Horowitz, and M. Levoy, "Synthetic Aperture Focusing using a Shear-Warp Factorization of the Viewing Transform," in *IEEE Conference on Computer Vision and Pattern Recognition - Workshops*, June 2005.

[10] Y. Lu, C. Guo, J. Qiu, P. Liu, and T. Ikenaga, "Low Complexity Homography Matrix Based SIFT for Real-Time 2D Rigid Object Tracking," in *International Conference on Wireless Communications Networking and Mobile Computing*, Sept 2010.

[11] Y. Cheng, "Real-time Surface Slope Estimation by Homography Alignment for Spacecraft Safe Landing," in *IEEE International Conference on Robotics and Automation*, May 2010, pp. 2280–2286.

[12] T. Wada, X. Wu, S. Tokai, and T. Matsuyama, "Homography Based Parallel Volume Intersection: Toward Real-Time Volume Reconstruction Using Active Cameras," in *IEEE International Workshop on Computer Architectures for Machine Perception*, 2000, pp. 331–339.

[13] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, New York, NY, USA, 2011.

[14] V. Popovic, H. Afshari, A. Schmid, and Y. Leblebici, "Real-time Implementation of Gaussian Image Blending in a Spherical Light Field Camera," in *Proceedings of IEEE International Conference on Industrial Technology*, February 2013, pp. 1173–1178.

[15] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 3rd ed. Berlin, Germany: Springer-Verlag, 2007.

Bibliography

[1] S.K. Nayar. Computational Cameras: Approaches, Benefits and Limits. Technical report, Computer Vision Laboratory, Columbia University, Jan 2011.

[2] Joaquim Salvi, Jordi Pages, and Joan Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4):827 – 849, 2004. doi: 10.1016/j.patcog.2003.10.002.

[3] Omer Cogal. *A Miniaturized Insect Eye Inspired Multi-camera Real-time Panoramic Imaging System.* PhD thesis, Swiss Federal Institute of Technology (EPFL), 2015.

[4] R. Szeliski and H-Y. Shum. Creating Full View Panoramic Image Mosaics and Environment Maps. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 251–258, New York, NY, USA, 1997. ACM. ISBN 0-89791-896-7. doi: http://dx.doi.org/10.1145/258734.258861.

[5] Matthew Brown and David Lowe. Automatic Panoramic Image Stitching Using Invariant Features. *International Journal of Computer Vision*, 74(1):59–73, August 2007.

[6] S. Peleg and J. Herman. Panoramic Mosaics by Manifold Projection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 338 –343, San Juan, Puerto Rico, June 1997. doi: 10.1109/CVPR.1997.609346.

[7] Yingen Xiong and Kari Pulli. Mask-based image blending and its applications on mobile devices. In *SPIE Multispectral Image Processing and Pattern Recognition (MIPPR)*, volume 7498, 2009. doi: 10.1117/12.832379.

[8] Anat Levin, Assaf Zomet, Shmuel Peleg, and Yair Weiss. Seamless image stitching in the gradient domain. In *Computer Vision - ECCV 2004*, volume 3024 of *Lecture Notes in Computer Science*, pages 377–389. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-24673-2_31.

[9] Patrick Perez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003. doi: 10.1145/882262.882269.

[10] R. Szeliski, M. Uyttendaele, and D. Steedly. Fast Poisson Blending using Multi-Splines. In *IEEE International Conference on Computational Photography (ICCP)*, 2011. doi: 10.1109/ICCPHOT.2011.5753119.

[11] J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum. Drag-and-drop pasting. In *ACM SIGGRAPH*, pages 631–637, 2006.

[12] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski. Coordinates for instant image cloning. *ACM Trans. Graph*, 28(3):1–9, 2009.

[13] P. Burt and E. Adelson. A Multiresolution Spline with Application to Image Mosaics. *ACM Trans. Graph.*, 2(4):217–236, October 1983. doi: 10.1145/245.247.

[14] M-S. Su, W-L. Hwang, and K-Y. Cheng. Variational Calculus Approach to Multiresolution Image Mosaic. In *Proceedings of International Conference on Image Processing*, volume 2, pages 245 –248, Oct. 2001. doi: 10.1109/ICIP.2001.958470.

[15] Christian Richardt, Yael Pritch, Henning Zimmer, and Alexander Sorkine-Hornung. Megastereo: Constructing High-Resolution Stereo Panoramas. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[16] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH 2001*, pages 341–346, 2001.

[17] Aseem Agarwala, Mira Dontcheva, Maneesh Agrawala, Steven Drucker, Alex Colburn, Brian Curless, David Salesin, and Michael Cohen. Interactive Digital Photomontage. *ACM Trans. Graph.*, 23(3):294–302, August 2004. doi: 10.1145/1015706.1015718.

[18] Yingen Xiong and Kari Pulli. Fast Panorama Stitching for High-Quality Panoramic Images on Mobile Phones. *IEEE Transactions on Consumer Electronics*, 56(2):298–306, 2010.

[19] Marc Levoy and Pat Hanrahan. Light Field Rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ISBN 0-89791-746-4. doi: http://doi.acm.org/10.1145/237170.237199.

[20] S.K. Nayar and A. Karmarkar. 360 x 360 Mosaics. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 388–395, Jun 2000.

[21] Karen B. Sarachik. Characterising an Indoor Environment with a Mobile Robot and Uncalibrated Stereo. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 984–989, 1989. doi: 10.1109/ROBOT.1989.100109.

[22] Heung-Yeung Shum and Li-Wei He. Rendering with Concentric Mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 299–306, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-48560-5. doi: 10.1145/311535.311573.

[23] Sudipta N. Sinha. Pan-Tilt-Zoom (PTZ) Camera. In Katsushi Ikeuchi, editor, *Computer Vision*, pages 581–586. Springer US, 2014. doi: 10.1007/978-0-387-31439-6_496.

[24] A.N. Belbachir, M. Mayerhofer, D. Matolin, and J. Colineau. Real-time 360° Panoramic Views Using BiCa360, the Fast Rotating Dynamic Vision Sensor to up to 10 Rotations per Sec. In *Proceedings of IEEE International Conference on Circuits and Systems*, pages 727–730, May 2012. doi: 10.1109/ISCAS.2012.6272139.

[25] A.N. Belbachir, R Pflugfelder, and Roman Gmeiner. A Neuromorphic Smart Camera for Real-time 360° Distortion-free Panoramas. In *Proceedings of IEEE International Conference on Distributed Smart Cameras*, pages 221–226, 2010.

[26] S. Baker and S.K. Nayar. A Theory of Catadioptric Image Formation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 35–42, Jan 1998.

[27] S.K. Nayar. Catadioptric Omnidirectional Camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 482–488, Jun 1997.

[28] Shree K Nayar and Venkata Peri. Folded Catadioptric Cameras. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 217–223, 1999.

[29] D. Taylor. Virtual Camera Movement: The Way of the Future? *American Cinematographer*, 77(8):93–100, 1996.

[30] O. Cogal, A. Akin, K. Seyid, V. Popovic, A. Schmid, and Y. Leblebici. A New Omni-Directional Multi-Camera System for High Resolution Surveillance. In *Proceeding of SPIE Defense and Security Symposium*, Baltimore, Maryland, USA, 2014. doi: 10.1117/12.2049698.

[31] K. Seyid, V. Popovic, O. Cogal, A. Akin, H. Afshari, A. Schmid, and Y. Leblebici. A Real-Time Multiaperture Omnidirectional Visual Sensor Based on an Interconnected Network of Smart Cameras. *Circuits and Systems for Video Technology, IEEE Transactions on*, 25 (2):314–324, February 2015. ISSN 1051-8215. doi: 10.1109/TCSVT.2014.2355713.

[32] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy. High Performance Imaging Using Large Camera Arrays. *ACM Trans. Graph.*, 24:765–776, July 2005. doi: 10.1145/1073204.1073259.

[33] Peter Rander, P. J. Narayanan, and Takeo Kanade. Virtualized Reality: Constructing Time-Varying Virtual Worlds From Real World Events. In *Proceedings of IEEE Visualization '97*, pages 277–284, October 1997.

[34] Cha Zhang and Tsuhan Chen. A Self-Reconfigurable Camera Array. In *Eurographics Symposium on Rendering*, pages 243–254, 2004.

[35] Wai-Kwan Tang, Tien-Tsin Wong, and Pheng-Ann Heng. A System for Real-Time Panorama Generation and Display in Tele-immersive Applications. *IEEE Transactions on Multimedia*, 7(2):280–292, April 2005.

[36] A. Majumder, WB. Seales, M. Gopi, and H. Fuchs. Immersive teleconferencing: a new algorithm to generate seamless panoramic video imagery. In *Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 169–178, 1999.

[37] Ulrich Neumann, Thomas Pintaric, and Albert Rizzo. Immersive Panoramic Video. In *Proceedings of the Eighth ACM International Conference on Multimedia*, MULTIMEDIA '00, pages 493–494, New York, NY, USA, 2000. ACM. ISBN 1-58113-198-4. doi: 10.1145/354384.376408. URL http://doi.acm.org/10.1145/354384.376408.

[38] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver. Google Street View: Capturing the World at Street Level. *Computer*, 43(6): 32–38, June 2010. doi: 10.1109/MC.2010.170.

[39] O. Schreer, I. Feldmann, C. Weissig, P. Kauff, and R. Schafer. Ultrahigh-Resolution Panoramic Imaging for Format-Agnostic Video Production. *Proceedings of the IEEE*, 101 (1):99–114, Jan 2013. doi: 10.1109/JPROC.2012.2193850.

[40] Yuan Xu, Qinghai Zhou, Liwei Gong, Mingcheng Zhu, Xiaohong Ding, and R.K.F. Teng. High-Speed Simultaneous Image Distortion Correction Transformations for a Multicamera Cylindrical Panorama Real-time Video System Using FPGA. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(6):1061–1069, June 2014. doi: 10.1109/TCSVT.2013.2290576.

[41] Seitz. Roundshot. http://www.roundshot.com. Accessed on Oct. 24, 2015.

[42] Kogeto. Jo. http://kogeto.com/jo.html. Accessed on Oct. 24, 2015.

[43] Pointgrey. Ladybug. https://www.ptgrey.com/360-degree-spherical-camera-systems. Accessed on Oct. 24, 2015.

[44] Fullview. http://www.fullview.com. Accessed on Oct. 24, 2015.

[45] Ricoh. THETA. https://theta360.com/. Accessed on Oct. 24, 2015.

[46] Jason C. Yang, Matthew Everett, Chris Buehler, and Leonard McMillan. A Real-Time Distributed Light Field Camera. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 77–86, 2002. ISBN 1-58113-534-3.

[47] Pelican Imaging. http://pelicanimaging.com. Accessed on Oct. 24, 2015.

[48] Lytro. http://lytro.com. Accessed on Oct. 24, 2015.

[49] Raytrix. http://raytrix.de. Accessed on Oct. 24, 2015.

[50] D. J. Brady, M. E. Gehm, R. A. Stack, D. L. Marks, D. S. Kittle, D. R. Golish, E. M. Vera, and S. D. Feller. Multiscale Gigapixel Photography. *Nature*, 486(7403):386–389, June 2012.

[51] Oliver S Cossairt, Daniel Miau, and Shree K Nayar. Gigapixel Computational Imaging. In *Proceedings of IEEE International Conference on Computational Photography*, pages 1–8, 2011.

[52] Y. M. Song, Y. Xie, V. Malyarchuk, J. Xiao, I. Jung, K.-J. Choi, Z. Liu, H. Park, C. Lu, R.-H. Kim, R. Li, K. B. Crozier, Y. Huang, and J. A. Rogers. Digital cameras with designs inspired by the arthropod eye. *Nature*, 497(7447):95–99, May 2013. doi: 10.1038/nature12083.

[53] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Cengage Learning, 3rd edition, 2008.

[54] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2nd edition, 2004.

[55] Vladan Popovic, Hossein Afshari, Alexandre Schmid, and Yusuf Leblebici. Real-time Implementation of Gaussian Image Blending in a Spherical Light Field Camera. In *Proceedings of IEEE International Conference on Industrial Technology*, pages 1173–1178, February 2013. doi: 10.1109/ICIT.2013.6505839.

[56] J. Bouget. Camera calibration toolbox for MATLAB. http://www.vision.caltech.edu/bouguetj/calib_doc. Accessed on Oct. 20, 2015.

[57] Sing Bing Kang and Richard S. Weiss. Can We Calibrate a Camera Using an Image of a Flat, Textureless Lambertian Surface? In *Proceedings of the 6th European Conference on Computer Vision - Part II*, pages 640–653, 2000. ISBN 3-540-67686-4.

[58] Vladan Popovic, Kerem Seyid, Abdulkadir Akin, Omer Cogal, Hossein Afshari, Alexandre Schmid, and Yusuf Leblebici. Image Blending in a High Frame Rate FPGA-based Multi-Camera System. *Journal of Signal Processing Systems*, 76:169–184, 2014. doi: 10.1007/s11265-013-0858-8.

[59] H. Afshari, A. Akin, V. Popovic, A. Schmid, and Y. Leblebici. Real-Time FPGA Implementation of Linear Blending Vision Reconstruction Algorithm Using a Spherical Light Field Camera. In *IEEE Workshop on Signal Processing Systems*, 2012. doi: 10.1109/SiPS.2012.49.

[60] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, New York, NY, USA, 2011. doi: 10.1007/978-1-84882-935-0.

[61] Vladan Popovic and Yusuf Leblebici. FIR filters for hardware-based real-time multi-band image blending. In *Proc. SPIE 9400, Real-Time Image and Video Processing, 94000D*, San Francisco, CA, USA, February 2015. doi: 10.1117/12.2078889.

[62] A. Goshtasby. Multi-exposure Image Dataset. http://www.imgfsr.com/. Accessed on Feb. 12, 2014.

[63] P. Marziliano, F. Dufaux, S. Winkler, and T. Ebrahimi. A No-Reference Perceptual Blur Metric. In *Proceedings of International Conference on Image Processing*, volume 3, pages 57–60, 2002. doi: 10.1109/ICIP.2002.1038902.

[64] C.S. Xydeas and V. Petrović. Objective image fusion performance measure. *Electronics Letters*, 36(4):308 –309, February 2000. doi: 10.1049/el:20000267.

[65] A. Mittal, R. Soundararajan, and A. C. Bovik. Making a completely blind image quality analyzer. *IEEE Signal Processing Letters*, 22(3):209–212, March 2013.

[66] J. Tierney, C. Rader, and B. Gold. A digital frequency synthesizer. *IEEE Transactions on Audio and Electroacoustics*, 19(1):48–57, March 1971.

[67] J. E. Volder. The CORDIC Trigonometric Computing Technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, September 1959.

[68] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer-Verlag, Berlin, Germany, 3rd edition, 2007.

[69] S. F. Anderson, J. G. Earle, R. E Goldschmidt, and D. M. Powers. The IBM System/360 model 91: Floating-point Execution Unit. *IBM Journal of Research and Development*, 11 (1):34–53, January 1967.

[70] Jianping Zhou. Getting the Most Out of Your Image-Processing Pipeline. Technical report, Texas Instruments, October 2007.

[71] Deuk Hyun Park, Hyoung Seok Ko, Jae Gon Kim, and Jun Dong Cho. Real Time Rectification Using Differentially Encoded Lookup Table. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, pages 47:1–47:4, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0571-6.

[72] C. Vancea and S. Nedevschi. LUT-based Image Rectification Module Implemented in FPGA. In *IEEE International Conference on Intelligent Computer Communication and Processing*, pages 147–154, 2007. doi: 10.1109/ICCP.2007.4352154.

[73] A. Akin, I. Baz, L.M. Gaemperle, A. Schmid, and Y. Leblebici. Compressed Look-Up-Table Based Real-Time Rectification Hardware. In *IFIP/IEEE 21st Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, pages 272–277, Oct 2013. doi: 10.1109/VLSI-SoC.2013.6673288.

[74] Vladan Popovic and Yusuf Leblebici. A Low-power 490 MPixels/s Hardware Accelerator for Pyramidal Decomposition of Images. In *Proceedings of IEEE International Conference on Image Processing (ICIP)*, Quebec City, Quebec, Canada, September 2015.

[75] Gyanesh Chander, Brian L. Markham, and Dennis L. Helder. Summary of current radiometric calibration coefficients for Landsat MSS, TM, ETM+, and EO-1 ALI sensors. *Remote Sensing of Environment*, 113(5):893–903, May 2009.

[76] Julia H. Jungmann, Luke MacAleese, Jan Visser, Marc J. J. Vrakking, and Ron M. A. Heeren. High Dynamic Range Bio-Molecular Ion Microscopy with the Timepix Detector. *Analytical Chemistry*, 83(20):7888–7894, 2011.

[77] Christian Bloch. *The HDRI Handbook 2.0: High Dynamic Range Imaging for Photographers and CG Artists*. Rocky Nook, 2013.

[78] S. Mann and R. W. Picard. On Being 'Undigital' with Digital Cameras: Extending Dynamic Range by Combining Differently Exposed Pictures. In *Proceedings of IS&T*, pages 442–448, 1995.

[79] Paul E. Debevec and Jitendra Malik. Recovering High Dynamic Range Radiance Maps from Photographs. In *ACM SIGGRAPH 97*, pages 369–378, New York, NY, USA, 1997. ISBN 0-89791-896-7.

[80] Greg Ward. *Graphics Gems II*, chapter Real Pixels, pages 80–83. Academic Press, San Diego, CA, USA, 1991.

[81] T. Mitsunaga and S.K. Nayar. Radiometric Self Calibration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 374–380, June 1999.

[82] Sumanta N. Pattanaik, Erik Reinhard, Greg Ward, and Paul E. Debevec. *High Dynamic Range Imaging - Acquisition, Display, and Image-Based Lighting*. Morgan Kaufmann, 2005.

[83] Mark A. Robertson, Sean Borman, and Robert L. Stevenson. Estimation-theoretic approach to dynamic range enhancement using multiple exposures. *Journal of Electronic Imaging*, 12(2):219–228, April 2003.

[84] Miguel Granados, Boris Ajdin, Michael Wand, Christian Theobalt, Hans-Peter Seidel, and Hendrik P. A. Lensch. Optimal HDR reconstruction with linear digital cameras. In *Proceedings of 23rd IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 215–222, 2010.

[85] Samuel W. Hasinoff, Frédo Durand, and William T. Freeman. Noise-Optimal Capture for High Dynamic Range Photography. In *Proceedings of 23rd IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 553–560, 2010.

[86] Tom Mertens, J. Kautz, and F. Van Reeth. Exposure Fusion. In *Pacific Conf. on Computer Graphics and Applications*, pages 382–390, 2007. doi: 10.1109/PG.2007.17.

[87] Amina Saleem, Azeddine Beghdadi, and Boualem Boashash. Image fusion-based contrast enhancement. *EURASIP Journal on Image and Video Processing*, 2012(10):1–17, 2012. doi: 10.1186/1687-5281-2012-10.

[88] Antonio Martinez-Sanchez, Carlos Fernandez, Pedro J. Navarro, and Andres Iborra. A novel method to increase linlog cmos sensors' performance in high dynamic range scenarios. *Sensors*, 11(9):8412–8429, 2011. doi: 10.3390/s110908412.

[89] Gregory Ward, Holly Rushmeier, and Christine Piatko. A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes. *IEEE Trans. Vis. Comput. Graphics*, 3(4):291–306, October 1997. doi: 10.1109/2945.646233.

[90] Sumanta N. Pattanaik, Jack Tumblin, Hector Yee, and Donald P. Greenberg. Time-dependent visual adaptation for fast realistic image display. In *ACM SIGGRAPH 00*, pages 47–54, New York, NY, USA, 2000. ISBN 1-58113-208-5. doi: 10.1145/344779.344810.

[91] F. Drago, K. Myszkowski, T. Annen, and N. Chiba. Adaptive Logarithmic Mapping For Displaying High Contrast Scenes. *Computer Graphics Forum*, 22(3):419–426, 2003. doi: 10.1111/1467-8659.00689.

[92] RafałMantiuk, Scott Daly, and Louis Kerofsky. Display Adaptive Tone Mapping. *ACM Trans. Graph.*, 27(3):68:1–68:10, August 2008.

[93] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic Tone Reproduction for Digital Images. *ACM Trans. Graph.*, 21(3):267–276, July 2002. doi: 10.1145/566654.566575.

[94] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient Domain High Dynamic Range Compression. *ACM Trans. Graph.*, 21(3):249–256, July 2002. doi: 10.1145/566654.566573.

[95] Frédo Durand and Julie Dorsey. Fast Bilateral Filtering for the Display of High-Dynamic-Range Images. *ACM Trans. Graph.*, 21(3):257–266, July 2002. doi: 10.1145/566654.566574.

[96] Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High Dynamic Range Video. *ACM Trans. Graph.*, 22(3):319–325, July 2003. doi: 10.1145/882262.882270.

[97] Nima Khademi Kalantari, Eli Shechtman, Connelly Barnes, Soheil Darabi, Dan B Goldman, and Pradeep Sen. Patch-based High Dynamic Range Video. *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH Asia 2013)*, 32(6):202, 2013.

[98] M. Gupta, D. Iso, and S.K. Nayar. Fibonacci Exposure Bracketing for High Dynamic Range Imaging. In *IEEE International Conference on Computer Vision (ICCV)*, 2013.

[99] Michael D. Tocci, Chris Kiser, Nora Tocci, and Pradeep Sen. A Versatile HDR Video Production System. *ACM Trans. Graph.*, 30(4):41:1–41:10, July 2011. doi: 10.1145/2010324.1964936.

[100] Joel Kronander, Stefan Gustavson, Gerhard Bonnet, and Jonas Unger. Unified HDR reconstruction from raw CFA data. In *Proceedings of IEEE International Conference on Computational Photography*, 2013.

[101] Vikas Ramachandra, Matthias Zwicker, and Truong Nguyen. HDR Imaging From Differently Exposed Multiview Videos. In *IEEE 3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video*, pages 85–88, 2008.

[102] T. Portz, L. Zhang, and H. Jiang. Random Coded Sampling for High-Speed HDR Video. In *IEEE International Conference on Computational Photography (ICCP)*, 2013. doi: 10.1109/ICCPhot.2013.6528308.

[103] Firas Hassan and Joan E. Carletta. An FPGA-based architecture for a local tone-mapping operator. *Journal of Real-Time Image Processing*, 2(4):293–308, 2007. doi: 10.1007/s11554-007-0056-7.

[104] Lavanya Vytla, Firas Hassan, and Joan E. Carletta. A real-time implementation of gradient domain high dynamic range compression using a local poisson solver. *Journal of Real-Time Image Processing*, 8(2):153–167, 2013. doi: 10.1007/s11554-011-0198-5.

[105] Pierre-Jean Lapray, B. Heyrman, Matthieu Rosse, and D. Ginhac. HDR-ARtiSt: High Dynamic Range Advanced Real-time Imaging System. In *IEEE International Symposium on Circuits and Systems*, pages 1428–1431, 2012. doi: 10.1109/ISCAS.2012.6271513.

[106] Pierre-Jean Lapray, Barthelemy Heyrman, and Dominique Ginhac. Hdr-artist: an adaptive real-time smart camera for high dynamic range imaging. *Journal of Real-Time Image Processing*, pages 1–16, 2014. doi: 10.1007/s11554-013-0393-7.

[107] Ahmet Oguz Akyuz. High dynamic range imaging pipeline on the GPU. *Journal of Real-Time Image Processing*, 10:273–287, 2012. doi: 10.1007/s11554-012-0270-9.

[108] Mohamed Akil, Thierry Grandpierre, and Laurent Perroton. Real-time dynamic tone-mapping operator on GPU. *Journal of Real-Time Image Processing*, 7(3):165–172, 2012. doi: 10.1007/s11554-011-0196-7.

[109] Marcos Slomp and Manuel M. Oliveira. Real-Time Photographic Local Tone Reproduction Using Summed-Area Tables. In *Computer Graphics International*, pages 82–91, Istanbul, Turkey, June 2008.

[110] H. Afshari, V. Popovic, T. Tasci, A. Schmid, and Y. Leblebici. A Spherical Multi-camera System with Real-time Omnidirectional Video Acquisition Capability. *IEEE Transactions on Consumer Electronics*, 58(4):1110–1118, November 2012. doi: 10.1109/TCE.2012.6414975.

[111] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2nd edition, 2000.

[112] Akiko Yoshida, Volker Blanz, Karol Myszkowski, and Hans-Peter Seidel. Perceptual Evaluation of Tone Mapping Operators with Real-World Scenes. In *SPIE Human Vision & Electronic Imaging X*, pages 192–203, 2005. doi: 10.1117/12.587782.

[113] V. Popovic, E. Pignat, and Y. Leblebici. Performance Optimization and FPGA Implementation of Real-Time Tone Mapping. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(10):803–807, October 2014. ISSN 1549-7747. doi: 10.1109/TCSII.2014.2345306.

[114] K. Kiratiratanapruk and S. Siddhichai. Vehicle detection and tracking for traffic monitoring system. In *TENCON 2006. 2006 IEEE Region 10 Conference*, pages 1–4, Nov 2006. doi: 10.1109/TENCON.2006.343888.

[115] Kyungnam Kim and LarryS. Davis. Object detection and tracking for intelligent video surveillance. In Weisi Lin, Dacheng Tao, Janusz Kacprzyk, Zhu Li, Ebroul Izquierdo, and Haohong Wang, editors, *Multimedia Analysis, Processing and Communications*, volume 346 of *Studies in Computational Intelligence*, pages 265–288. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-19550-1. doi: 10.1007/978-3-642-19551-8_9. URL http://dx.doi.org/10.1007/978-3-642-19551-8_9.

[116] Chris McClanahan. History and evolution of gpu architecture, 2010.

[117] M.J. Misic, D.M. Djurdjevic, and M.V. Tomasevic. Evolution and trends in gpu computing. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 289–294, May 2012.

[118] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):1–45, December 2006. ISSN 0360-0300. doi: 10.1145/1177352.1177355. URL http://doi.acm.org/10.1145/1177352.1177355.

[119] M. Piccardi. Background subtraction techniques: a review. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 4, pages 3099–3104 vol.4, Oct 2004. doi: 10.1109/ICSMC.2004.1400815.

[120] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, Dec 2008. doi: 10.1109/ICPR.2008.4760998.

[121] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4:51–52, 2001.

[122] Akintola Kolawole and Alireza Tavakkoli. Robust foreground detection in videos using adaptive color histogram thresholding and shadow removal. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, volume 6939 of *Lecture Notes in Computer Science*, pages 496–505. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24030-0. doi: 10.1007/978-3-642-24031-7_50. URL http://dx.doi.org/10.1007/978-3-642-24031-7_50.

List of Publications

Book Chapters

- K. Seyid, Ö. Cogal, V. Popovic, H. Afshari, A. Schmid, and Y. Leblebici. Real-Time Omnidirectional Imaging System with Interconnected Network of Cameras. *VLSI-SoC: Internet of Things Foundations*, Springer Publishing Group, p. 170-197, 2015.

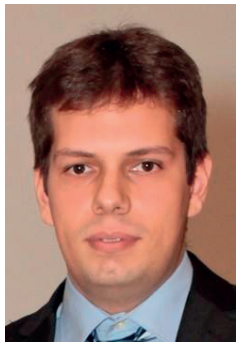
Journals

- K. Seyid, V. Popovic, Ö. Cogal, A. Akin, H. Afshari, A. Schmid, and Y. Leblebici. A Real-time Multi-aperture Omnidirectional Visual Sensor Based on Interconnected Network of Smart Cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(2):314-324, 2015.
- N. Katic, V. Popovic, R. Cojbasic, A. Schmid and Y. Leblebici. A Relative Imaging CMOS Image Sensor for High Dynamic Range and High Frame-Rate Machine Vision Imaging Applications. *IEEE Sensors Journal*, 15(7):4121-4129, 2015.
- V. Popovic, K. Seyid, E. Pignat, Ö. Cogal and Y. Leblebici. Multi-Camera Platform for Panoramic Real-Time HDR Video Construction and Rendering. *Journal of Real-Time Image Processing*, 2014.
- V. Popovic, E. Pignat and Y. Leblebici. Performance Optimization and FPGA Implementation of Real-Time Tone Mapping. *IEEE Transactions on Circuits and Systems II: Express Briefs,* 61(10):803-807, October 2014.
- V. Popovic, K. Seyid, A. Akin, Ö. Cogal, H. Afshari, A. Schmid, and Y. Leblebici. Image Blending in a High Frame Rate FPGA-based Multi-Camera System. *Journal of Signal Processing Systems for Signal, Image and Video Technology,* 76(2):169-184, 2014. **(invited)**
- H. Afshari, V. Popovic, T. Tasci, A. Schmid and Y. Leblebici. A Spherical Multi-camera System with Real-time Omnidirectional Video Acquisition Capability. *IEEE Transactions on Consumer Electronics,* 58(4):1110-1118, 2012.

Conferences

- V. Popovic and Y. Leblebici. A Low-Power 490 MPixels/s Hardware Accelerator for Pyramidal Decomposition of Images. *IEEE International Conference on Image Processing (ICIP)*, Quebec City, Canada, 2015.
- V. Popovic and Y. Leblebici. FIR filters for hardware-based real-time multi-band image blending. *SPIE Electronic Imaging Conference*, San Francisco, CA, USA, 2015.
- F. Depraz, V. Popovic, B. Ott, P. Wellig and Y. Leblebici. Real-time object detection and tracking in omni-directional surveillance using GPU. *SPIE Conference on Defense and Security Europe,* Toulouse, France, 2015.
- S. Ergünay, K. Seyid, V. Popovic and Y. Leblebici. A Novel Hybrid Architecture for Real-Time Omnidirectional Image Reconstruction. *IEEE International Conference on Distributed Smart Cameras,* Seville, Spain, 2015. **(invited)**
- V. Popovic and Y. Leblebici. Reconfigurable Forward Homography Estimation System for Real-Time Applications. *IEEE International Conference on Very Large Scale Integration (VLSI-SoC),* Playa del Carmen, Mexico, 2014.
- L. M. Gaemperle, K. Seyid, V. Popovic and Y. Leblebici. An Immersive Telepresence System using a Real-Time Omnidirectional Camera and a Virtual Reality Head-Mounted Display. *IEEE International Symposium on Multimedia,* Taichung, Taiwan, 2014.
- Ö. Cogal, A. Akin, K. Seyid, V. Popovic and A. Schmid et al. A New Omni-Directional Multi-Camera System for High Resolution Surveillance. *SPIE Defense and Security Symposium,* Baltimore, MD, USA 2014.
- V. Popovic, H. Afshari, A. Schmid and Y. Leblebici. Real-time Implementation of Gaussian Image Blending in a Spherical Light Field Camera. *IEEE International Conference on Industrial Technology,* Cape Town, South Africa 2013.
- V. Popovic, K. Seyid, A. Schmid and Y. Leblebici. Real-Time Hardware Implementation of Multi-Resolution Image Blending. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP),* Vancouver, BC, Canada, 2013.
- Ö. Cogal, V. Popovic and Y. Leblebici. Spherical Panorama Construction using Multi Sensor Registration Priors and Its Real-time Hardware. *IEEE International Symposium on Multimedia,* Anaheim, CA, USA, 2013.
- H. Afshari, A. Akin, V. Popovic, A. Schmid and Y. Leblebici. Real-time FPGA implementation of linear blending vision reconstruction algorithm using a spherical light field camera. *IEEE Workshop on Signal Processing Systems,* Québec City, Québec, Canada, 2012. **(Best Student Paper Award)**

Vladan Popović



Chemin de Grande-Rive 7,
1007 Lausanne
+41 (0)78 9275312
Date of Birth: 14.11.1986
Nationality: Serbian
vladan.popovic@epfl.ch
http://people.epfl.ch/vladan.popovic

EDUCATION

2011 – 2015 **PhD candidate at Microelectronic Systems Laboratory, Ecole Polytechnique Federale de Lausanne (EPFL), Switzerland**
- supervisor: Prof. Yusuf Leblebici

2015 **Visiting Student Researcher, Stanford University**
- Stanford Computational Imaging Group, Prof. Gordon Wetzstein

2009 - 2011 **MSc in Electrical and Electronics Engineering, EPFL** (GPA: 5.64/6.00)
- Microelectronics orientation with Minor in Space Technology
- Ranked 5th out of 55 students

2005 - 2009 **BSc in Electrical Engineering, Faculty of Electrical Engineering, Belgrade** (GPA: 9.50/10.00)

AWARDS

- **Grand Prix** at Laval Virtual 2015
- **Environment & Health** prize at Laval Virtual 2015
- **Best Student Paper Award** at IEEE Workshop on Signal Processing Systems (SIPS 2012), in Québec City, Québec, Canada, 17-19 October 2012.
- **Logitech prize** for Master thesis, for its outstanding creativity, innovation, pragmatism and economic feasibility
- EPFL Excellence Scholarship

RESEARCH INTERESTS

- Multi-camera systems
- Computational photography
- Real-time image and video processing systems
- Embedded systems design
- Heterogeneous systems

WORK EXPERIENCE

2010-2015 **Teaching assistant at EPFL, Lausanne**
- Signal Processing for Communications, Hardware Systems Modelling, Digital Systems Design

2009 **Teaching assistant at Faculty of Electrical Engineering, Belgrade**
- Fundamentals of Electrical Engineering (responsibility of grading).

2008 **Intern at TeleGroup d.o.o., Belgrade** (2 months summer internship)
- PHP/MySQL web development

COMPUTER SKILLS

Languages: Matlab, C/C++, Java, VHDL, Verilog, Python, x86 Assembler

Software: Windows, Linux, MS Visual Studio, Eclipse, Xilinx ISE/XPS/Vivado, ModelSim, Quartus, NIOS IDE, Cadence, Synopsis

PROFESSIONAL TRAINING

- Venture Challenge, Venture Lab 2014
- Cryptographic Engineering, MEAD Education FP7, 2014

PROFESSIONAL SERVICE

- Reviewer for IEEE Transactions on VLSI
- Session Chair at IEEE International Conference on Image Processing (ICIP) 2015

LANGUAGES

Serbian	**English**	**French**
Native language	Fluent (C2), Cambridge Certificate of Advanced English	Intermediate (B1/B2)

SYNOPSIS OF RESEARCH

Research in the field of computational photography increased vastly in the last decade. Apart from new photographic capabilities, which extend beyond the ones of the traditional camera, performance plays an important role in the imaging system. In the undergoing project, I designed and successfully tested three omnidirectional light field video camera platforms, with the appropriate trade-off between the system's performance (bandwidth, frame rate) and image quality (resolution, visual perception). The image reconstruction algorithms are adapted for FPGA implementation, which is the core processing unit of our embedded camera system.